Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik I



# Models and Algorithms for Online Exploration and Search

**Dissertation**

Zur Erlangung des Doktorgrades (Dr. rer. nat.)
der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

**Thomas Kamphans**

Bonn, 2005

Angefertigt mit der Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

Gutachter: Prof. Dr. Rolf Klein, Universität Bonn
Prof. Dr. Alejandro López-Ortiz, University of Waterloo, Canada

Tag der mündlichen Prüfung: 04.04.2006

Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn
`http://hss.ulb.uni-bonn.de/diss_online`
elektronisch publiziert.

Erscheinungsjahr: 2006

**Abstract**

This work considers some algorithmic aspects of exploration and search, two tasks that arise, for example, in the field of motion planning for autonomous mobile robots. We assume that the environment is not known to the robot in advance, so we deal with *online* algorithms.

First, we consider a special kind of environments that we call *cellular environments*, where the robot's surrounding is subdivided by an integer grid. The robot's task is to visit every cell in this grid at least once. We distinguish between *simple* grid polygons (i. e., polygons with no obstacles inside) and general grid polygons. We show that no online exploration strategy is able to achieve a competitive factor better than $\frac{7}{6}$ for simple grid polygons and better than 2 for general grid polygons. That is, the path of an online exploration strategy is in the worst case at least $\frac{7}{6}$ times (2 times, respectively) longer than the optimal path that was computed with full knowledge of the environment. For both cases we develop exploration strategies and show upper bounds on their performance. More precisely, for environments without obstacles we provide a strategy that produces tours of length $S \leq C+\frac{1}{2}E-3$, and for environments with obstacles we provide a strategy that is bound by $S \leq C + \frac{1}{2}E + 3H + W_{\mathrm{cw}} - 2$, where $C$ denotes the number of cells—the area—, $E$ denotes the number of boundary edges—the perimeter—, $H$ is the number of obstacles, and $W_{\mathrm{cw}}$ is a measure for the sinuosity of the given environment. Moreover, we show that the strategy for simple grid polygons is $\frac{4}{3}$-competitive; that is, the path generated by our strategy is never longer than $\frac{4}{3}$ times the optimal path.

Second, we consider search tasks with error-prone robots and give performance results that take the robot's errors into account. The first search task is to leave an unknown environment using the well-known *Pledge* algorithm. We give sufficient conditions that ensure a successful application with an error-prone robot. The second task is the search for a door in a wall (or a point on a line). We show that a robot that is not aware of making errors is able to find its goal, if its error is not greater than 33 per cent. Further, we give an optimal-competitive strategy that takes the maximal error into account, and generalize our result to searching on $m$ rays.

Last, we examine a new cost measure for search tasks, the *search ratio*. The quality of a search path is determined by a worst-case target point—a point that maximizes among all target points, $t$, the ratio between the length of the searcher's path up to $t$ and the shortest path to $t$. An *optimal search path* has the minimal search ratio among all search paths in the given environment. The *optimal search ratio*—the search ratio of the optimal search path—is an appropriate measure for the *searchability* of an environment. We give a general framework for approximating a path with optimal search ratio, and apply this framework to simple polygons and grid polygons. Further, we show that no constant-competitive approximation is possible for polygons with holes.

What I most of all regret
Is not what I did
But all the things that I've left undone
(Justin Sullivan)

**Acknowledgments**

First of all, I would like to thank my advisor, Prof. Dr. Rolf Klein, for giving me the opportunity to write this thesis and plenty of valuable thoughts and advices, Prof. Dr. Alejandro López-Ortiz for accepting to be the second referee for this work, and Dr. Elmar Langetepe for a great deal of helpful discussions and bright ideas.

Further, I would like to thank my coauthors and colleagues, Annette Ebbers-Baumann, Andrea Eubeler, Prof. Dr. Rudolf Fleischer, Ansgar Grüne, Dr. Christian Icking and Gerhard Trippen—it has been a pleasure to work with you. I'm also much grateful to our student workers, Jens Behley, Ulrich Handel and Wolfgang Meiswinkel, for their great help with the GridRobot applet, Christian Moll for some proofreadings, and Mariele Knepper for her assistance with administrative tasks.

Above all, warmest thanks to my parents, Friedrich and Brigitte Kamphans for their constant encouragement and continuous support, to my brothers, Stefan and Matthias, and to Dorthe Lübbert for many things beyond this thesis.

# Contents

# Chapter 1

# Introduction

Designing appropriate models is a crucial task in many sciences. This ranges from concrete models such as scaled reproductions of city parts used by architects to determine how a new building fits into the existing surrounding; over simplifications—for instance, electrical engineers use equivalent circuit diagrams for complex components such as transistors to simplify the calculation of voltages and currents in circuits—to highly abstracted models that map the *real world* into *computable* terms and play an import role in computer science and mathematics. Perhaps, the most frequently used model for real-world matters are graphs, which are used to describe city maps, railroad or computer networks, relationships between persons and many other things, in a way that can be stored in known data structures and handled with a large number of known algorithms.

All models have in common a certain degree of abstraction and often of simplification. For example, scaled city models do not show fine details of the houses, because that does not serve the purpose the models are made for. And a transistor used for simple on–off switching can be described with a model that is much simpler than the equivalent circuit diagram for a transistor serving as HiFi amplifier. The challenge is to find models that are simple enough, but not too simple, so that it is easy to map a given setting into the model and it is easy to work with the model, whereas the model still serves its purpose.

### Models in Robot Motion Planning

This work addresses *exploration* and *search*, two tasks that arise from the field of *robot motion planning*, where we want to compute trajectories for autonomous mobile robots—vehicles equipped with some kind of intelligence so they can move around without being steered by a human operator—such as the robot shown in Figure 1.1. But, although we often talk about robots and have robots as primary application in mind, all presented algorithms may be applied by agents of any kind, this may be a person mowing a lawn

Figure 1.1: A mobile robot (Activmedia Pioneer P2-AT) equipped with a laser scanner (Sick).

(in Chapter 2), a person searching a lost item (in Chapter 4), a walker (in Chapter 3), or even—as Kao et al. introduce a certain search problem—a cow searching for a feedlot.[1] Thus, we use the terms *robot*, *searcher*, *explorer*, or *agent* synonymously.

To be able to solve such motion planning tasks we have to formalize them. That is, we have to design appropriate models for the robot and the robot's environment. Further, we want to evaluate the quality of our motion planning algorithms; thus, we also have to discuss possible ways to model the costs incurred by an algorithm.

In the following paragraphs, we give attention to commonly used models for robot motion planning. Further, we briefly review some notions we use in this work. For more theoretical background we refer the reader to books on computational geometry or geometric modeling, for example O'Rourke [150, 151], Abramowski and Müller [2], Klein [114, 115], or de Berg et al. [39]. See also the book of Schwartz and Yap [167], as well as the handbooks by Goodman and O'Rourke [67], and Sack and Urrutia [159].

**Environment**

First, we have to find an appropriate model for the robot's environment. In most applications there are areas in which the robot can move around—the *free space*—and areas that are impenetrable for the robot. The latter areas are called *obstacles*. The free space may be unbounded or bounded in the case of a robot moving inside a closed room.

In many applications we want to calculate a path for a robot moving in a two-dimensional environment, typically a floor plan. In this case the obstacles and—in the case of a bounded free space—the robot's work area are usually modeled by simple polygons. A *polygon* is a region that is enclosed

[1] Provided that there will ever be a cow that is able to read this work.

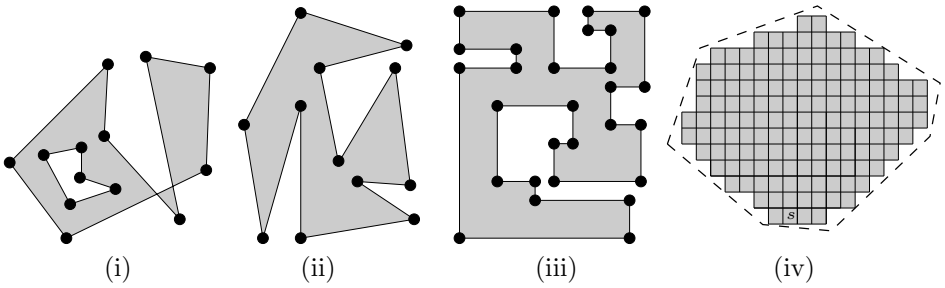



Figure 1.2: Several types of polygons: (i) polygon with hole, (ii) simple polygon, (iii) rectilinear, simple polygon, (iv) grid polygon.

by a closed polygonal chain (i. e., a set of concatenated line segments). If the polygon is topologically equivalent to a disk; that is, the polygon is enclosed by a single, nonintersecting polygonal chain, we call the polygon *simple*. Note that a simple polygon does not contain any other polygon (*hole*). A polygon whose edges meet with internal angles of either $\frac{\pi}{2}$ or $\frac{3}{2}\pi$ is called a *rectilinear* or *orthogonal polygon*, see Figure 1.2. Environments consisting of a set of obstacles given by simple polygons are called *polygonal scenes*.

The robot's environment may be modeled using also closed curves, or approximated by an integer grid, see Figure 1.2(iv). We discuss the latter approach in Chapter 2.

Sometimes it is possible to abstract from the geometry of the real environment and consider only connections between parts of the surrounding, such as paths between crossings and dead-ends in a classic example of a labyrinth. In this case, we may use graphs to model the environment. Further, we may give grid polygons, see Figure 1.2(iv), as *grid graphs* that consist of one vertex per square and edges between neighboring squares.

**Robot**

There are many kinds of robots having different sizes, computational abilities, sensors, and drive mechanisms. Thus, we have to decide whether our robot model has to reproduce the robot's dimensions, or whether it is sufficient to approximate the robot's shape, maybe, by a circle. Many algorithms in robot motion planning completely abstract from the robot's measures and deal only with point-shaped robots.

Another important issue is the sensor model. Basically, we distinguish between *blind* robots (i. e., robots that are equipped with touch sensors that allow only the very close environment to be detected by the robot), and robots that have *vision*, such as a sonar or a laser scanner. In the idealistic case, a vision sensor provides the full *visibility polygon*; that is, the set of all points in the environment that are visible from the robot's current position, see Figure 1.3(i). The *visibility*, in turn, depends on the type of the environment. For example, a point, $p$, inside a simple polygon, $P$, is visible

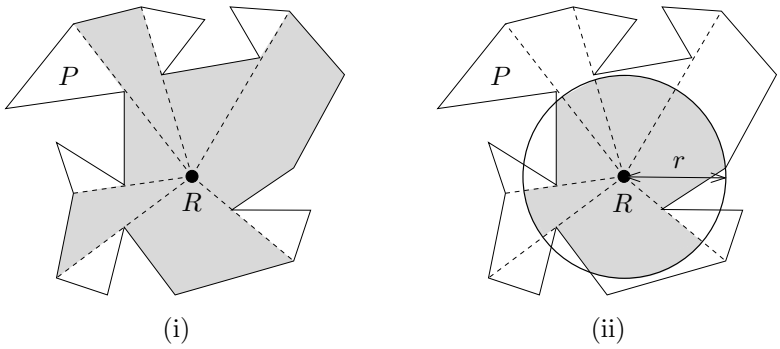



Figure 1.3: (i) The visibility polygon (shaded) of $P$ with respect to the robot's current position, $R$, (ii) limited visibility polygon.

from another point, $q \in P$, if the line segment from $p$ to $q$ is completely contained in $P$. We may also consider a limited vision sensor; in this case, the robot gets the intersection of the visibility polygon with a circle whose radius is determined by the range of the scanner, see Figure 1.3(ii).

Further, we have to regard the computational abilities, essentially the memory size—is the robot able to store a map of the whole environment, or is the memory limited to a few words?—, and the motion abilities. The accuracy of both the input data and the motion is also a relevant item. Theorists often assume that robots are error free. On the other hand, practitioners often give only statistically or empirically obtained correctness results and performance guarantees (e.g. [12, 13, 119, 120, 127, 186, 189]). There are basically three approaches to deal with errors: The first objective is to *reduce errors*; either by reducing odometry errors (e.g., Chong and Kleeman [35], Borenstein and Feng [21, 22]) or by avoiding faulty data by using more reliable input (e.g., preferring angular measures over distance measures; see Lumelsky and Tiwari [138], Angluin et al. [7], Demaine et al. [40], or LaValle et al. [126]). Dudek et al. [47] presented an exploration strategy for a group of robots, where the moving robot uses the other robots as landmarks. The second approach is to *tolerate errors* and show that the strategy is robust under certain types of errors (e.g., Noborio et al. [144, 145, 146], López-Ortiz and Schuierer [130]). Another method is to *detect errors and react appropriately* (e.g., Byrne et al. [28], Zelinsky [190], or Stentz [174, 175]). We consider robots with errors in Chapter 3.

**Costs**

Given two algorithms, how can we determine which one is better suited? To decide this question, we need appropriate models for the quality of an algorithm. A commonly used model for the costs of an algorithm is to account its need for resources—usually computing time and memory allocation—in terms of the input size using the well-known order notation; see, for exam-

ple, Knuth [117]. Sometimes, the size of the output is considered, too. Such algorithms are called *output sensitive*.

Keeping computing time and memory allocation low is often a secondary goal in robotics. Mobile robots are powered by rechargeable batteries, and as the power consumption of the robot's motors dominates the power consumption of the onboard computer, we are primarily interested in paths that are as short as possible. Sometimes, also other cost measures such as turning or scanning costs are considered.

In robot motion planning we often deal with algorithms that do not have all the information needed to compute an optimal solution, such as a robot moving in an unknown terrain. While the robot moves around in the terrain, it learns the environment by and by. This kind of algorithms is called *online algorithms*, in contrast to *offline algorithms* that compute the solution having the full information.

The *competitive ratio* is a commonly used performance measure for online algorithms. We compare the costs of an online algorithm with the costs of an optimal offline algorithm. If this ratio is bounded by a constant for arbitrary instances of the problem, we call the algorithm *competitive*. More precisely:

**Definition 1.1** Let ONL be an online algorithm. We call ONL *competitive* with factor $C$ (or $C$-competitive for short), if there exists a constant, $A$, so that for every possible input to ONL

$$|\mathrm{ONL}| \leq C \cdot |\mathrm{OPT}| + A$$

holds, where OPT denotes the optimal solution, and |ONL| and |OPT| the costs of ONL and OPT, respectively.

The constant $A$ in Definition 1.1 ensures a bounded ratio for certain start situations. Imagine a searcher located in the origin and searching for goal on the real line. The search strategy moves the searcher one unit to the right in the first step. Now, a malicious adversary reveals the goal at distance $\varepsilon > 0$ to the left of the searcher's start. Since $\varepsilon$ can be arbitrarily small, our competitive factor, $C$, goes to infinity. To avoid this, we define $A = 1$. Alternatively, we may introduce certain assumptions on the start situation; for example, we may require that the distance to the goal is at least 1.

The work of Sleator and Tarjan [173] concerning self-organizing lists and paging was one of the first using competitive analysis. Since then, online algorithms have been studied in many different areas, such as an online version of the traveling salesman problem (Ausiello et al. [10]), online leasing (El-Yaniv [50]), seat reservation (Boyar and Larsen [25]), or buying a *Bahncard*[2] (Fleischer [58]). See also the books by Fiat and Woeginger [57], and Borodin and El-Yaniv [23].

[2] A card for sales discount for the German railroad.

A more general understanding of competitivity is to introduce a function, $f(n)$, instead of the constant $C$, where $n$ denotes the size of the input. Thus, we are able to classify an algorithm, for example, as $\sqrt{n}$-competitive. However, in this work we use the term *competitive* as defined in Definition 1.1 (i. e., constant competitive) unless explicitly mentioned.

The competitive factor of a certain strategy gives us an *upper bound* for the competitive complexity of the considered problem; that is, we know that the problem cannot be harder, because we are already able to solve it with the given competitive factor. On the other hand, we may be able to give certain scenarios in which every possible strategy cannot be better than a proven factor. These settings are called *lower bounds*. If the competitive factor of a strategy exactly matches the corresponding lower bound, we know that we can not improve the strategy—at least not in the competitive framework. Thus, we call such a strategy *optimal competitive*.

In the usual competitive framework we compare an online algorithm to the optimal solution. Of course, there are other possibilities, such as the *excess distance ratio*, see Berman [15], that compares the online strategy to certain dimensions of the environment. Lumelsky et al. [137, 135] analyzed their solutions in comparison to the sum of the obstacles' perimeters. We use a similar approach in Chapter 2. Another model is the *search ratio*, see Koutsoupias et al. [118], Fleischer et al. [59], and Chapter 4.

### Robot Motion Planning Tasks

Path-planning strategies for mobile autonomous robots in different settings have attracted a lot of researchers. The settings differ in the model for the robot and the environment, online and offline settings, and, of course, the robot's task. Basically, four types of path planning tasks have been investigated, namely navigation, searching, exploration, and localization.

In the *navigation* task, the robot has to find a path to a target—not necessarily a point—whose location is known to the robot. In contrary, *searching* means that the target is unknown to the robot. Note that navigation in the offline setting amounts to finding a (shortest) obstacle-avoiding path from the start to the target.

*Exploration* refers to the task of finding a path, such that every point in the environment is seen from at least one point on the path. The details depend on the type of environment and the robot. For example, a robot equipped with an unlimited vision sensor moving in a simple polygon, $P$, has to find a path, $\pi$, so that for every point $p \in P$ there is at least one point $p' \in \pi$ so that $p$ is visible from $p'$; that is, the line segment from $p$ to $p'$ is completely contained in $P$.

At first view, search and exploration seem to be closely related. After all, every search strategy has to inspect the whole environment; otherwise, the target could be located in an unseen part and the search fails. Therefore,

every search strategy is also an exploration strategy. The main difference is that an online exploration strategy competes merely against the optimal offline exploration, whereas a search strategy is compared to the shortest path from the start to the target, which may be much shorter than an optimal exploration path. In Chapter 4 we discuss this difference in more detail. In particular, we show that there is, anyway, a close relation between exploration and search.

Some authors distinguish between exploration—seeing every point in the environment, possibly from far away, e. g. for map-making purposes—and *covering*, where every part in the environment has to be visited by the robot, maybe, to accomplish some work like lawn-mowing. However, for robots without vision both tasks are the same. Another slight variation is to inspect only the obstacles' boundaries instead of the whole environment—sometimes, this task is called *mapping*. Similar to searching, the online tasks are identical, but an offline mapping path may be shorter than an offline exploration path, because the former is allowed to skip hidden, but obstacle-free areas.

In the *localization* setting, the environment is known in advance, but the robot does not know its current position inside the map; imagine a cleaning device that is positioned somewhere in an office-building and powered on.

In the following, we briefly review some previous results in algorithmic motion planning. For a general overview on theoretical online motion planning see the survey articles by Rao et al. [158], Icking and Klein [90], Berman [15], Trippen [184], and Icking et al. [88].

We concentrate on motion planning in a geometric perspective, and disregard other—no less interesting—techniques such as the potential field method, where the motion planning problem is modeled by electrostatic-like attraction and repulsion, or the probabilistic roadmap approach, see, e. g., Overmars [152], Švestka and Overmars [177], Kavraki et al. [110], and the survey by Overmars [153]. These topics are addressed in the surveys by Hwang and Ahuja [85]; Halperin, Kavraki, and Latombe [73, 74]; and the comprehensive book by Latombe [123]. Further, we restrict ourself to planning tasks for a single robot.

Needless to mention, robot navigation tasks have also been studied from a rather practical point of view by numerous authors such as Rao et al. [157], VanderHeide and Rao [186], Lee and Recce [127], Kuipers and Byun [119, 120], Batalin and Sukhatme [12, 13], Taylor and Kriegman [183]—just to list a few of them. See also the book by Choset et al. [37], the forthcoming book by LaValle [125], or the survey by Choset [36] on recent results on covering.

**Navigation**

Among the first navigation strategies were the Bug algorithms by Lumelsky and Stepanov [137] for finding a target with a point-shaped robot using

a touch sensor and a compass directed towards the target. Many Bug-like strategies have been proposed since then, such as Sankaranarayanan and Vidyasagar [160], or Rajko and LaValle [155]. A Bug-like algorithm was also used for the Mars Rover project, see Laubach and Burdick [124]. Bug strategies for robots with a (limited) vision sensor were introduced by Lumelsky and Skewis [136].

Navigation in polygonal scenes and graphs was studied by Papadimitriou and Yannakakis [154]. They showed that no strategy can achieve a constant competitive factor in a polygonal scene if the obstacles have an unbounded aspect ratio. For scenes with square obstacles they gave a lower bound of $\frac{3}{2}$, and suggested strategies that achieve this ratio asymptotically. Blum, Raghavan, and Schieber [19] considered—among other things—the *wall problem*, where the target is an infinite line, and presented an optimal $O(\sqrt{n})$-competitive algorithm for this problem. Berman et al. [16] gave a randomized $O(n^{\frac{4}{9}} \log n)$-competitive navigation strategy.

The offline navigation task amounts to compute a (shortest) path from $s$ to $t$. For point-shaped robots in a polygonal scene this is possible in time $O(n \log n)$ (Hershberger and Suri [80]), inside a simple polygon in time $O(\log n + k)$ after an $O(n)$-preprocessing, where $k$ denotes the number of segments on the shortest path (Guibas and Hershberger [71]). Computing a shortest path in a scene with polyhedral obstacles is known to be NP-hard, see Canny and Reif [29]. Path planning for non–point-shaped robots were considered, for example, by Icking et al. [96] for line segments; Ó'Dúnlaing and Yap [149] for discs; and Kedem, Sharir, and Toledo [111, 112] for convex robots. See also the surveys by Schwartz and Sharir [165, 166], Sharir [170], and Mitchell [141, 142], as well as the book by Agarwal and Sharir [171].

**Searching**

Searching has been studied broadly in the context of game theory. Two players, a searcher and a hider, compete against each other. The searcher moves around in the environment and tries to find the hider as soon as possible, whereas the objective of the hider is to maximize the search time. Search games date back to the works of Koopmann in 1946 and Bellmann in 1956, see the books of Gal [64], and Alpern and Gal [6] for a comprehensive overview on search games. Claude Shannon [168] constructed a machine that moved an electrical "finger" through a labyrinth to find a target. Although his search strategy is rather simple, the implementation of the algorithm was very remarkable at that time. Labyrinth searching was also considered in the context of automata theory; see for instance Blum and Kozen [20]. Obviously, labyrinths can be modeled as graphs; thus, labyrinth searching amounts to searching in a graph. This was studied by Tarry and Tremaux back in the nineteenth century; their algorithms led to the well-known depth-first search (DFS) and breadth-first search (BFS) graph traversals.

The presumably most simple search task is the search for a point on an infinite line. The searcher may be a robot searching for a door in a long wall, or a walker searching for a bridge across a river. Beck and Newman [14], Gal [64], and independently Baeza-Yates, Culberson, and Rawlins [11] studied this problem. Both works introduced the *doubling strategy* and showed that an optimal competitive factor of 9 is achievable. The doubling strategy is a fundamental paradigm for other search problems. For a more detailed description of the doubling strategy see Section 3.2.

Searching on the line was generalized to searching on $m$ rays emanating from a single source, see Gal [64] and Baeza-Yates et al. [11]. Many other variants were discussed since then, for example $m$-ray searching with restricted goal distance (Hipke et al. [82], Langetepe [122], López-Ortiz and Schuierer [162, 131]), $m$-ray searching with additional turn costs (Demaine et al. [41]), parallel $m$-ray searching (Kao et al. [108], Hammar et al. [76], López-Ortiz and Schuierer [133]) or randomized searching (Schuierer [163], Kao et al. [109]). Furthermore, some of the problems were again rediscovered by Jaillet et al. [98].

Whereas there is no constant-competitive strategy for searching in an arbitrary simple polygon, see Figure 4.1 on page 108, Klein [113] introduced a special kind of simple polygons, the *streets*, that allow constant-competitive searching. Icking, Klein, and Langetepe [93], and independently Schuierer and Semrau [164] presented a strategy with an optimal competitive factor of $\sqrt{2}$, see also Icking et al. [94], Icking [86], and Langetepe [122]. López-Ortiz and Schuierer [130] gave search strategy for streets that is robust under small navigational errors. The works mentioned so far assumed that the start and the target are the two points that are used to define a street. Bröcker and López-Ortiz [26] considered searching in streets with arbitrary start and target points.

Kleinberg [116] gave a $O(k)$-competitive search strategy for rectilinear simple polygons, where $k$ denotes the number of essential cuts.[3] Searching in arbitrary simple polygons was considered by Schuierer [161], and Klein [114, 115]; their strategies are $O(n)$-competitive for a polygon with $n$ vertices. Searching in polygonal scenes was considered, for example, by Kalyanasundaram and Pruhs [99].

Because there is no constant-competitive search strategy in trees and graphs, Koutsoupias, Papadimitriou, and Yannakakis [118] introduced the *search ratio* of a tree or graph as the best achievable competitive factor for a search in the given environment. We attend to the search ratio in Chapter 4.

In a special case of searching, we just want to leave an unknown scene; that is, we search for the boundary of the scene. This problem can be solved using the algorithm of Pledge, see Abelson and diSessa [1], and Hemmerling [79]. We consider the Pledge algorithm in Section 3.1.

[3] See Section 4.2.2 for the definition of essential cuts.

Among other search tasks are the search for the kernel of a polygon (Icking et al. [92], Langetepe [122]), searching on a lattice (Baeza-Yates et al. [11], López-Ortiz and Sweet [134]), searching in a star polygon (López-Ortiz and Schuierer [132]), searching for a line (Gal [64] and Baeza-Yates et al. [11]) or a ray (Eubeler et al. [54]) in the plane. Usually, the path length or the search time is used to measure the quality of a search strategy. Fekete, Klein, and Nüchter [56] considered the number of scans to measure the costs. More search problems are presented, for example, in López-Ortiz [128] and Alpern and Gal [6]. See also the book by Ahlswede and Wegener [3] on (nongeometric) search problems.

**Exploration**
The task of exploring an unknown simple polygon using a point-shaped robot equipped with an unlimited, error-free vision system[4] starting in a point, $s$, on the polygon's boundary was first considered by Deng, Kameda and Papadimitriou [42, 43]. Their strategy is optimal for rectilinear simple polygons with respect to the optimal path in the $L_1$-metric and $\sqrt{2}$-competitive with respect to the optimum in the $L_2$-metric. For nonrectilinear simple polygons they claimed a factor of 2016. With the same assumptions, Hoffmann, Icking, Klein, and Kriegel introduced a 133-competitive strategy [83] that was finally improved to a 26.5-competitive algorithm called *PolyExplore*, see [84]. We will briefly review these algorithms in Section 4.2.2. Albers, Kursawe, and Schuierer [5] showed a lower bound of $\Omega(\sqrt{n})$ for the exploration of polygons with holes. For the case that $s$ is an arbitrary point inside a rectilinear simple polygon, Kleinberg [116] gave a lower bound of $\frac{5}{4}$ and a randomized $\frac{5}{4}$-competitive exploration strategy.

The optimal offline exploration path in a simple polygon starting in a fixed point, $s$, on the polygon's boundary is also known as the shortest watchman route, and was first considered by Chin and Ntafos [32]. They provided an $O(n)$-algorithm for shortest watchman routes in rectilinear simple polygons with $n$ vertices. Some work has been done on shortest watchman routes, see [33, 179, 75, 182, 181]—some of them vainly tried to generalize the algorithm by Chin and Ntafos—, until Dror, Efrat, Lubiw and Mitchell [46] presented an $O(n^3 \log n)$-algorithm for shortest watchman routes in arbitrary simple polygons. Similar problems are the *floating* SWR (i. e., the shortest route without a fixed start point), see Carlsson, Jonsson, and Nilsson [31], and Tan [178]; or the shortest watchman *path* with different start- and end points (Carlsson and Jonsson [30]). Other variants are, for instance, zookeeper routes[5] (Chin and Ntafos [34], Bespamyatnikh [17]), safari routes[6]

[4] Thus, the full visibility polygon with respect to the robot's current position is provided.

[5] Given a simple polygon, $P$, a set, $\mathcal{P}$, of convex polygons inside $P$, and a start point; find the shortest route that touches each polygon from $\mathcal{P}$ but enters none of them.

[6] Basically the same as zookeeper routes, but it is allowed to enter the polygons in $\mathcal{P}$.

(Tan and Hirata [180]), aquarium keeper routes[7] (Czyzowicz et al. [38]), or robber routes[8] (Ntafos [147]).

Betke, Rivest, and Singh [18] introduced the *piecemeal exploration*, where the robot has to interrupt the exploration every now and then so as to return to the start point, for example, to refuel. They presented two constant-competitive strategies for the piecemeal exploration of grid graphs[9] with rectangular obstacles. In this case, exploration means that the robot has to visit of every node as well as every edge. Their result was generalized to arbitrary rectilinear obstacles by Albers, Kursawe, and Schuierer [5].

For the exploration of graphs see Kalyanasundaram and Pruhs [100], Albers and Henzinger [4], Deng and Papadimitriou [44], and Fleischer and Trippen [62]. Mapping was considered, for example, by Kalyanasundaram and Pruhs [99]. Lumelsky, Mukhopadhyay, and Sun [135] provided two algorithms for mapping unknown polygonal scenes, and analyzed their performance basically in terms of the obstacles' perimeters.

**Localization**

Guibas, Motwani, and Raghavan [72] and independently Bose, Lubiw, and Munro [24] considered the problem of finding the set of possible locations of a robot inside a known map based on the robot's visibility polygon. Moving the robot eliminates wrong guesses in the set of possible locations. Dudek, Romanik, and Whitesides [48] presented an optimal competitive strategy to find a path that leads to an uniquely determined location. Fleischer et al. [61] presented an optimal $O(\sqrt{n})$-competitive algorithm for the same problem in trees. Furthermore, Demaine, López-Ortiz and Munro [40] considered localization with help of landmarks. In the robotics community, localization is often solved using probablistic approaches; see, for example, Burgard et al. [27].

[7] The shortest route that visits every edge of a simple polygon.

[8] Given a simple polygon, $P$, a set, $\mathcal{T}$, of points inside $P$ (the threats), and a set, $\mathcal{S}$, of line segments on the boundary of $P$ (the sights); find the shortest route that sees at least one point of each line segment in $\mathcal{S}$, but is not seen from any point in $\mathcal{T}$.

[9] A graph with only axis-parallel, unit-sized edges, see Chapter 2.

### Overview of this Work

This work is organized as follows. In Chapter 2 we consider the exploration task for a simplified environment model: A robot without vision moves in a polygon that consists of square-shaped cells. We distinguish between environments with and without holes. For both settings we give lower bounds, suggest exploration algorithms, and analyze them in terms of the polygon's dimensions. For polygons without holes we also analyze the exploration strategy in the competitive framework. A preliminary version of the strategy for polygons with holes was presented at the 16th European Workshop on Computational Geometry (Euro-CG 2000) [87], see also [88]. The exploration of simple grid polygons was presented at the 11th International Computing and Combinatorics Conference (COCOON 2005) [89].

Chapter 3 deals with error-prone robots. First, we consider the Pledge algorithm and develop conditions that guarantee a success, even if the robot is erroneous. Amongst others we show that a robot using a compass with an accuracy of only $\pm\frac{\pi}{2}$ is still able to leave an unknown maze. Afterward, we analyze the usual doubling strategy for searching a point on a line with an erroneous robot, and give an optimal competitive search strategy for the error-prone case. Further, we study the search on $m$-rays. Preliminary versions of these topics were presented at the First Workshop on Approximation and Online Algorithms (WAOA 2003) [103], the 20th Euro-CG 2004 [104], and the Fourth International Workshop on Efficient and Experimental Algorithms (WEA 2005) [106]. See also the technical report [107].

A special technique for measuring search costs, the *search ratio*, is covered in Chapter 4. We give a general framework for approximating a path with optimal search ratio, and apply this framework to simple polygons. Further, we show that no constant-competitive approximation is possible for polygons with holes. Preliminary versions have been published in the abstracts of the 20th Euro-CG 2004 [60] and in proceedings of the 12th Annual European Symposium on Algorithms (ESA 2004) [59].

Some of the results presented in the chapters 2–4 also appeared in [102].

# Chapter 2

# Exploring Cellular Environments

The exploration of unknown environments is—as already mentioned in the introduction—one of the basic tasks of autonomous mobile robots. In this chapter, we introduce a quite simple model for the robot and its environment: The robot is short sighted, and the surrounding is subdivided by a rectangular integer grid. Thus, the robot moves in a cellular environment, similar to a chessboard or squared writing paper, see Figure 2.1. In spite of the very basic sensors, we assume that the robot is equipped with enough memory to store a map of visited cells.

Essentially, there are two motivations for using this model instead of a robot that moves in an arbitrary (simple) polygon and is equipped with an ideal vision system that provides the full visibility polygon:

- In practice, there is no ideal vision system. Even the range of realistic laser scanners is limited to a few meters, see, for example, [172] or [81]. Therefore, the robot has to move towards areas in farther distance to explore them. In our model, the fineness of the grid (i. e., the size of a single *cell* in the environment) is determined by the reliable range of the laser scanner.

- Service robots like lawn mowers or cleaning devices need to get close to the parts of the environment they want to visit. Moreover, robots of this kind have to be rather cheap to be accepted by customers. Hence, such robots are not equipped with an expensive vision system. In this setting, the size of the robot or its tool defines the size of a cell, and we subdivide the environment according to the cell size.

We call the set of all cells that can be reached by the robot a *grid polygon*, or polygon for short. The robot starts from a cell, $s$, inside the polygon and adjacent to polygon's boundary. The robot's sensors provide the information

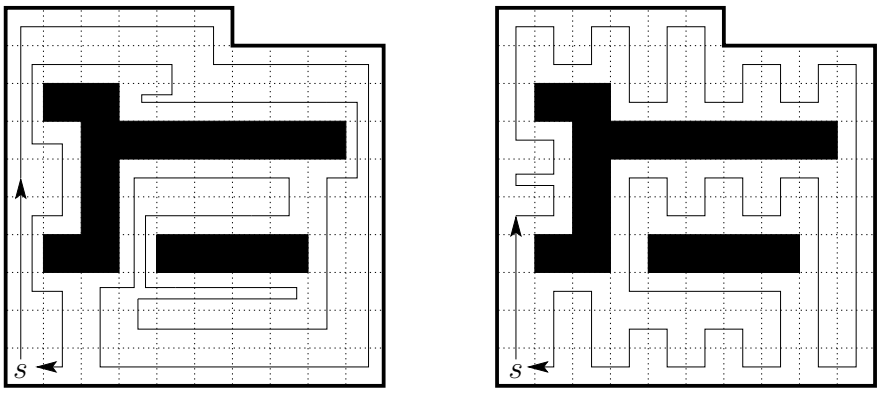



Figure 2.1: (i) An example exploration tour, (ii) a shortest TSP tour for the same polygon. The black cells show obstacles inside the polygon.

which of the four neighbors of the currently occupied cell do not belong to the polygon and which ones do. The robot can enter the latter cells. The task is to visit every cell inside the polygon and to return to the start cell.[1] The example in Figure 2.1(i) shows a tour that visits each cell at least once, but some cells even more. We are interested in a short exploration tour, so we would like to keep the number of additional cell visits small.

The equivalent offline problem—in this setting, the environment is known to the robot—, results in the construction of a shortest traveling salesman tour on the polygon cells, see Figure 2.1(ii). For polygons with obstacles, the problem of finding such a minimum length tour is known to be NP-hard, see Itai et al. [97]. There are $1+\varepsilon$ approximation schemes by Grigni et al. [69], Arora [9], and Mitchell [140], and a $\frac{53}{40}$ approximation by Arkin et al. [8].

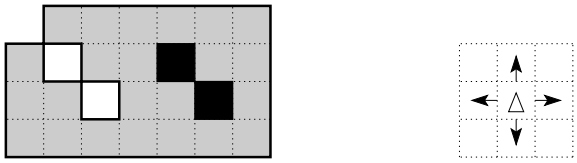
In polygons without obstacles, the complexity of constructing a minimum length tour offline seems to be open. Ntafos [148] and Arkin et al. [8] showed how to approximate the minimum length tour with factors of $\frac{4}{3}$ and $\frac{6}{5}$, respectively. Umans and Lenhart [185] provided an $O(C^4)$ algorithm for deciding if there exists a Hamiltonian cycle, that is, a tour that visits each of the $C$ cells of a polygon *exactly* once. For the related problem of Hamiltonian paths (i. e., a path with different start and end positions), Everett [55] presented a polynomial algorithm for certain grid graphs.

We are interested in the online version of the cell exploration problem. The task of exploring a grid polygon *with* holes was independently considered by Gabriely and Rimon [63]. They introduce a somehow artificial robot model by distinguishing between the robot and its tool, see Section 2.4.2. This model allows a smart analysis yielding an upper bound of $C+B$, where

[1] Sometimes, this task is also called *covering*.

$C$ denotes the number of cells and $B$ the number of boundary cells. However, this bound is generally larger than our bound, except for corridors of width 1, in which both bounds are the same. This may justify our more detailed analysis of the strategy. The piecemeal exploration of grid graphs[2] was studied by Betke et al. [18] and Albers et al. [5]. Note that their objective is to visit every node *and* every edge, whereas we require a complete coverage of only the cells. Subdividing the robot's environment into grid cells is used also in the robotics community, see, for example, Moravec and Elfes [143], and Elfes [51].

In the following, we give some lower bounds on the problem, see Section 2.1. Further, we consider the exploration of simple grid polygons in Section 2.2 and the case of polygons with holes in Section 2.3. But first, we want to give a more detailed description of our environments.



(i) (ii)

Figure 2.2: (i) Polygon with 23 cells, 38 edges and one(!) hole (black cells), (ii) the robot can determine which of the 4 adjacent cells are free, and enter an adjacent free cell.

**Definition 2.1** A *cell* is a basic block in our environment, defined by a tuple $(x, y) \in \mathbb{N}^2$. A cell is either *free* and can be visited by the robot, or *blocked* (i. e., unaccessible for the robot).[3] We call two cells $c_1 = (x_1, y_1), c_2 = (x_2, y_2)$ *adjacent* or *neighboring*, if they share a common edge (i. e., if $|x_1 - x_2| + |y_1 - y_2| = 1$ holds), and *touching*, if they share a common edge or corner.

A *path*, $\pi$, from a cell $s$ to a cell $t$ is a sequence of free cells $s = c_1, \ldots, c_n = t$ where $c_i$ and $c_{i+1}$ are adjacent for $i = 1, \ldots, n-1$. Let $|\pi|$ denote the length of $\pi$. We assume that the cells have unit size, so the length of the path is equal to the number of *steps* from cell to cell that the robot walks.

A *grid polygon*, $P$, is a connected set of free cells; that is, for every $c_1, c_2 \in P$ exists a path from $c_1$ to $c_2$ that lies completely in $P$.

We call a set of touching blocked cells that are completely surrounded by free cells an *obstacle* or *hole*, see Figure 2.2. Polygons without holes are called *simple polygons*.

[2]The grid graph corresponding to a grid polygon, $P$, consists of one node for every free cell in $P$. Two nodes are connected by an edge, if their corresponding cells are adjacent.

[3]In the following, we sometimes use the terms *free cells* and *cells* synonymously.

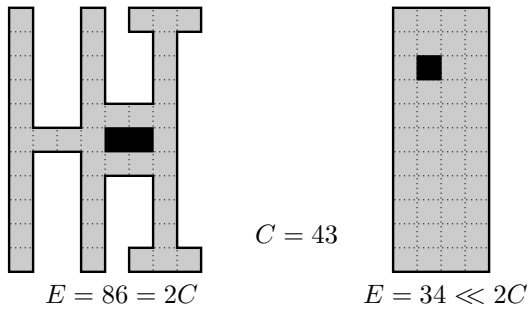



Figure 2.3: The perimeter, $E$, is used to distinguish between *thin* and *thick* environments.

We analyze the performance of an exploration strategy using some parameters of the grid polygon. In addition to the area, $C$, of a polygon we introduce the *perimeter*, $E$. $C$ is the number of free cells and $E$ is the total number of edges that appear between a free cell and a blocked cell, see, for example, Figure 2.2 or Figure 2.3. We use $E$ to distinguish between thin and thick environments, see Section 2.1. In Section 2.3.2 we introduce another parameter, the sinuosity $W_{\text{cw}}$, to distinguish between straight and twisted polygons.

## 2.1 Competitive Complexity

We are interested in an online exploration. In this setting, the environment is not known to the robot in advance. Thus, the first question is whether the robot is still able to approximate the optimum solution up to a constant factor in this setting. There is a quick and rather simple answer to this question:

**Theorem 2.2** *The competitive complexity of exploring an unknown cellular environment with obstacles is equal to 2.*

**Proof.** Even if the environment is unknown we can apply a simple depth-first search algorithm (DFS) to the grid graph. This results in a complete exploration in $2C-2$ steps. The shortest tour needs at least $C$ steps to visit all cells and to return to $s$, so DFS is competitive with a factor of 2.

On the other hand, 2 is also a lower bound for the competitive factor of any strategy. To prove this, we construct a special grid polygon depending on the behavior of the strategy. The start position, $s$, is located in a long corridor of width 1. We fix a large number, $Q$, and observe how the strategy explores this corridor. Two cases occur.

**Case 1:** The robot eventually returns to $s$ after walking at least $Q$ and at most $2Q$ steps. At this time, we close the corridor with two unvisited

cells, one at each end, see Figure 2.4(i). Let $R$ be the number of cells visited so far. The robot has already walked at least $2R-2$ steps and needs another $2R$ steps to visit the two remaining cells and to return to $s$, whereas the shortest tour needs only $2R$ steps to accomplish this task.

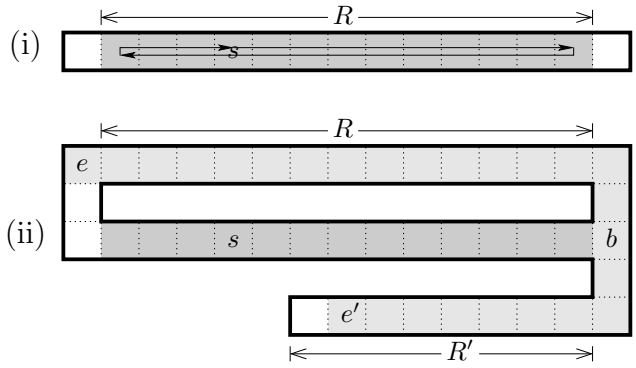



Figure 2.4: A lower bound of 2 for the exploration of grid polygons.

**Case 2:** In the remaining case the robot concentrates—more or less—on one end of the corridor. Let $R$ be the number of cells visited after $2Q$ steps. Now, we add a bifurcation at a cell $b$ immediately behind the farthest visited cell in the corridor, see Figure 2.4(ii). Two paths arise, which turn back and run parallel to the long corridor. If the robots returns to $s$ before exploring one of the two paths an argument analogous to case 1 applies. Otherwise, one of the two paths will eventually be explored up to the cell $e$ where it turns out that this corridor is connected to the other end of the first corridor. At this time, the other path is defined to be a dead end of length $R'$, which closes just one cell behind the last visited cell $e'$.

From $e$ the robot still has to walk to the other end of the corridor, to visit the dead end, and to return to $s$. Altogether, it will have walked at least four times the length of the corridor, $R$, plus four times the length of the dead end, $R'$. The optimal path needs only $2R + 2R'$, apart from a constant number of steps for the vertical segments.

In any case, the lower bound for the number of steps tends to 2 while $Q$ goes to infinity. □

We cannot apply Theorem 2.2 to simple polygons, because we used a polygon with a hole to show the lower bound. The following lower bound holds for simple polygons.

**Theorem 2.3** *Every strategy for the exploration of a simple grid polygon with $C$ cells needs at least $\frac{7}{6}C$ steps.*

**Proof.** We assume that the robot starts in a corner of the polygon, see Figure 2.5(i) where △ denotes the robot's position. Let us assume, the

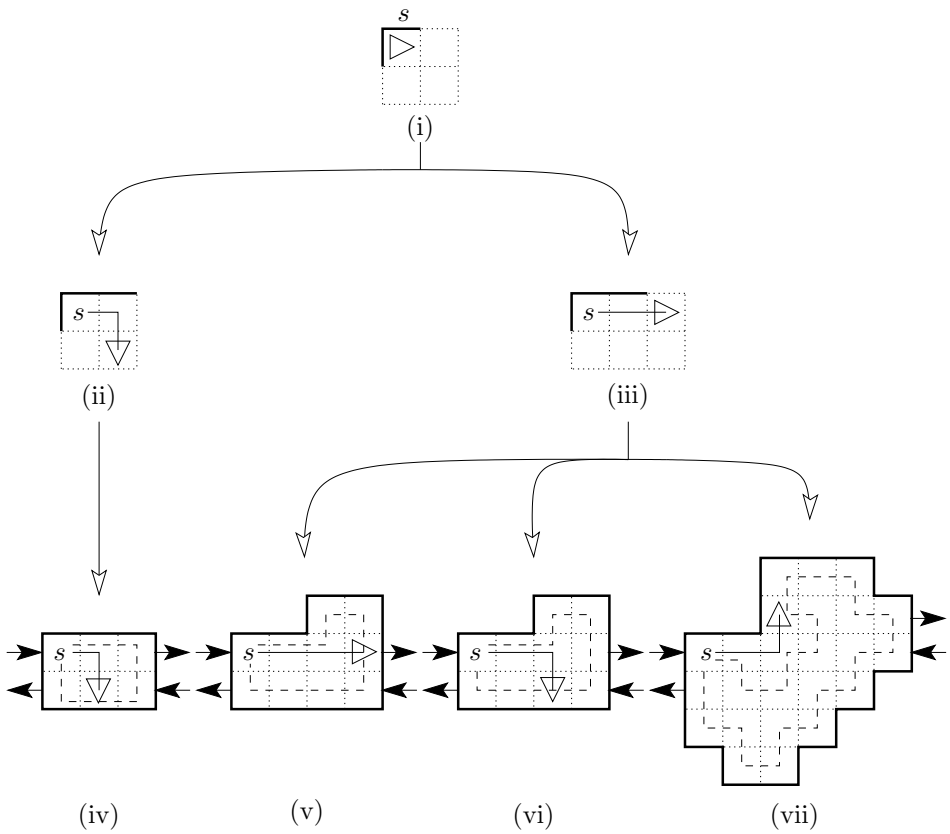



Figure 2.5: A lower bound for the exploration of simple polygons. The dashed lines show the optimal solution.

strategy decides to walk one step to the east—if the strategy walks to the south we use a mirrored construction. For the second step, the strategy has two possibilities: Either it leaves the wall with a step to the south, see Figure 2.5(ii), or it continues to follow the wall with a further step to the east, see Figure 2.5(iii). In the first case, we close the polygon as shown in Figure 2.5(iv). The robot needs at least 8 steps to explore this polygon, but the optimal strategy needs only 6 steps yielding a factor of $\frac{8}{6} \approx 1.3$. In the second case we proceed as follows. If the robot leaves the boundary, we close the polygon as shown in Figure 2.5(v) and (vi). The robot needs 12 step, but 10 steps are sufficient. In the most interesting case, the robot still follows the wall, see Figure 2.5(vii). In this case, the robot will need at least 28 steps to explore this polygon, whereas an optimal strategy needs only 24 steps. This leaves us with a factor of $\frac{28}{24} = \frac{7}{6} \approx 1.16$.

We can easily extend this pattern to build polygons of arbitrary size by repeating the preceding construction several times using the *entry* and *exit* cells denoted by the arrows in Figure 2.5(iv)–(vii). As soon as the robot leaves one block, it enters the start cell of the next block and the game starts

again; that is, we build the next block depending on the robot's behavior. Note that this construction cannot lead to overlapping polygons or polygons with holes, because the polygon always extends to the same direction. □

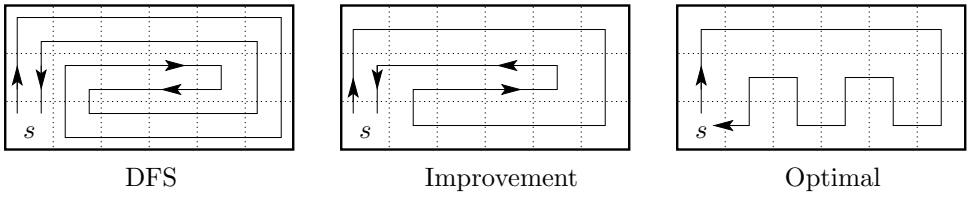



Figure 2.6: DFS is not the best possible strategy.

Even though we have seen in Theorem 2.2 that the simple DFS strategy already achieves the optimal competitive factor in polygons with holes, DFS is not the best possible exploration strategy! There is no reason to visit *each* cell twice just because this is required in some special situations like dead ends of width 1. Instead, a strategy should make use of wider areas, see Figure 2.6.

We use the perimeter, $E$, to distinguish between thin environments that have many corridors of width 1, and thick environments that have wider areas, see Figure 2.3 on page 16. In the following sections we present strategies that explore grid polygons using no more than roughly $C + \frac{1}{2}E$ steps. Since all cells in the environment have to be visited, $C$ is a lower bound on the number of steps that are needed to explore the whole polygon and to return to $s$.[4] Thus, $\approx \frac{1}{2}E$ is an upper bound for the number of additional cell visits. For thick environments, the value of $E$ is in $O(\sqrt{C})$, so that the number of additional cell visits is substantially smaller than the number of free cells. Only for polygons that do not contain any $2\times2$ square of free cells, $E$ achieves its maximum value of $2(C + 1)$, and our upper bound is equal to $2C - 2$, which is the cost of applying DFS. But in this case one cannot do better, because even the optimal offline strategy needs that number of steps. In other cases, our strategies are more efficient than DFS.

[4]More precisely, we need at least $C - 1$ steps to visit every cell, and at least 1 step to return to $s$.

## 2.2 Exploring Simple Polygons

We have seen in the previous section that a simple DFS traversal achieves a competitive factor of 2. Because the lower bound for simple grid polygons is substantially smaller, there may be a strategy that yields a better factor. Indeed, we can improve the DFS strategy. In this section, we give a precise description of DFS and present two improvements that lead to a $\frac{4}{3}$-competitive exploration strategy for simple polygons.

### 2.2.1 An Exploration Strategy

There are four possible directions—north, south, east and west—for the robot to move from one cell to an adjacent cell. We use the command *move(dir)* to execute the actual motion of the robot. The function *unexplored(dir)* returns true, if the cell in the given direction seen from the robot's current position is not yet visited, and false otherwise. For a given direction *dir*, *cw(dir)* denotes the direction turned 90° clockwise, *ccw(dir)* the direction turned 90° counterclockwise, and *reverse(dir)* the direction turned by 180°.

Using these basic commands, the simple DFS strategy can be implemented as shown in Algorithm 2.1. For every cell that is entered in direction *dir*, the robot tries to visit the adjacent cells in clockwise order, see the procedure *ExploreCell*. If the adjacent cell is still unexplored, the robot enters this cell, recursively calls *ExploreCell*, and walks back, see the procedure *ExploreStep*. Altogether, the polygon is explored following the *left-hand rule*: The robot proceeds from one unexplored cell to the next while the polygon's boundary or the explored cells are always to its left hand side.

Obviously, all cells are visited, because the graph is connected, and the whole path consists of $2C-2$ steps, because each cell—except for the start—is entered exactly once by the first *move* statement, and left exactly once by the second *move* statement in the procedure *ExploreStep*.

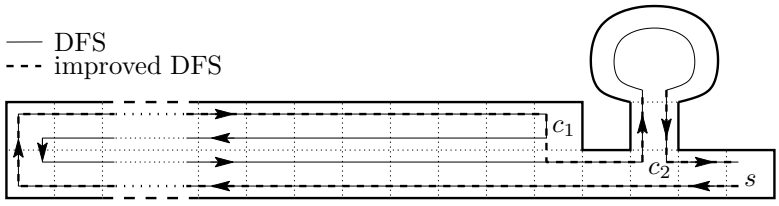



Figure 2.7: First improvement to DFS: Return directly to those cells that still have unexplored neighbors.

The first improvement to the simple DFS is to return directly to those cells that have unexplored neighbors. See, for example, Figure 2.7: After the robot has reached the cell $c_1$, DFS walks to $c_2$ through the completely

**Algorithm 2.1** DFS

```
DFS(P, start):
    Choose direction dir, so that reverse(dir) points to a blocked cell;
    ExploreCell(dir);

ExploreCell(dir):
    // Left-Hand Rule:
    ExploreStep(ccw(dir));
    ExploreStep(dir);
    ExploreStep(cw(dir));

ExploreStep(dir):
    if unexplored(dir) then
      move(dir);
      ExploreCell(dir);
      move(reverse(dir));
    end if
```

explored corridor of width 2. A more efficient return path walks on a shortest path from $c_1$ to $c_2$. Note that the robot can use for this shortest path only cells that are already known. With this modification, the robot's position might change between two calls of *ExploreStep*. Therefore, the procedure *ExploreCell* has to store the current position, and the robot has to walk on the shortest path to this cell, see the procedure *ExploreStep* in Algorithm 2.2. The function *unexplored(cell, dir)* returns true, if the cell in direction *dir* from *cell* is not yet visited.

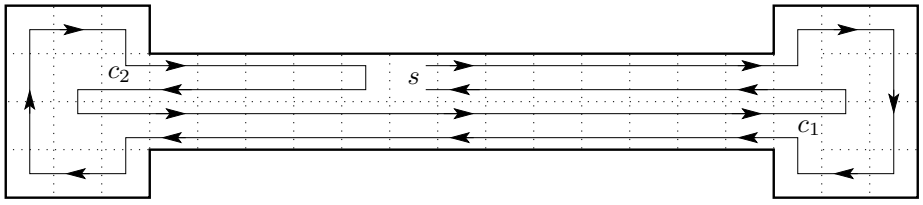



Figure 2.8: Second improvement to DFS: Detect polygon splits.

Now, observe the polygon shown in Figure 2.8. DFS completely surrounds the polygon, returns to $c_2$ and explores the left part of the polygon. After this, it walks to $c_1$ and explores the right part. Altogether, the robot walks four times through the narrow corridor. A more clever solution would explore the right part immediately after the first visit of $c_1$, and continue with the left part after this. This solution would walk only two times through the corridor in the middle! The cell $c_1$ has the property that the graph of

unvisited cells splits into two components after $c_1$ is explored. We call cells like this *split cells*. The second improvement to DFS is to recognize split cells and diverge from the left-hand rule when a split cell is detected. Essentially, we want to split the set of cells into several components, which are finished in the reversed order of their distances to the start cell. The detection and handling of split cells is specified in Section 2.2.2. Algorithm 2.2 resumes both improvements to DFS.

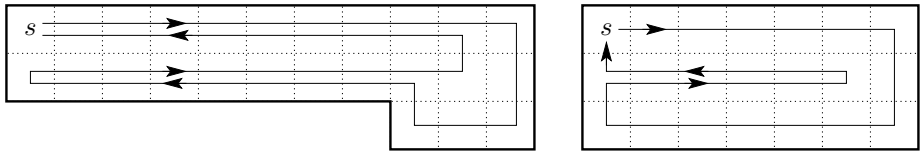



Figure 2.9: Straightforward strategies are not better than SmartDFS.

Note that the straightforward strategy *Visit all boundary cells and calculate the optimal offline path for the rest of the polygon* does not achieve a competitive factor better than 2. For example, in Figure 2.9(i) this strategy visits almost every boundary cell twice, whereas SmartDFS visits only one cell twice. Even if we extend the simple strategy to detect split cells while visiting the boundary cells, we can not achieve a factor better than $\frac{4}{3}$. A lower bound on the performace of this strategy is a corridor of width 3, see Figure 2.9(ii). Moreover, it is not known whether the offline strategy is NP-hard for simple polygons.

**Algorithm 2.2** SmartDFS

**SmartDFS(**$P$**,** *start***):**

Choose direction *dir* for the robot, so that reverse(*dir*) points to a blocked cell;
ExploreCell(*dir*);
Walk on the shortest path to the start cell;

**ExploreCell(**$dir$**):**

Mark the current cell with the number of the current layer;
*base* := current position;
**if** not isSplitCell(*base*) **then**
  // Left-Hand Rule:
  ExploreStep(*base*, ccw(*dir*));
  ExploreStep(*base*, *dir*);
  ExploreStep(*base*, cw(*dir*));
**else**
  // choose different order, see page 26 ff
  Determine the types of the components using the layer numbers of the surrounding cells;
  **if** No component of type III exists **then**
    Use the left-hand rule, but omit the first possible step.
  **else**
    Visit the component of type III at last.
  **end if**
**end if**

**ExploreStep(**$base$**,** $dir$**):**

**if** unexplored(*base*, *dir*) **then**
  Walk on shortest path using known cells to *base*;
  move(*dir*);
  ExploreCell(*dir*);
**end if**

### 2.2.2 The Analysis of SmartDFS

SmartDFS explores the polygon in layers: Beginning with the cells along the boundary, SmartDFS proceeds towards the interior of $P$. Let us number the single layers:

**Definition 2.4** Let $P$ be a (simple) grid polygon. The boundary cells of $P$ uniquely define the *first layer* of $P$. The polygon $P$ without its first layer is called the *1-offset* of $P$. The $\ell$th layer and the $\ell$-offset of $P$ are defined successively, see Figure 2.10.

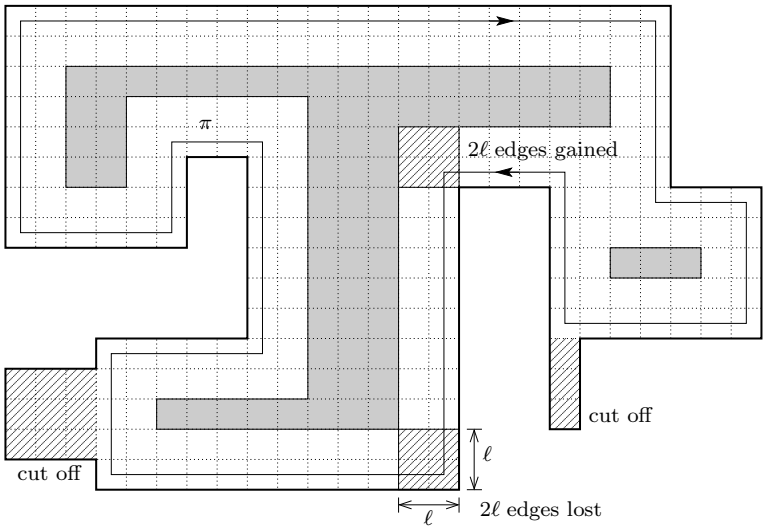



Figure 2.10: The 2-offset (shaded) of a grid polygon $P$.

Note that the $\ell$-offset of a polygon $P$ is not necessarily connected. Although the preceding definition is independent from any strategy, SmartDFS can determine a cell's layer when the cell is visited for the first time. We can define the $\ell$-offset in the same way for a polygon with holes, but the layer of a given cell can no longer be determined on the first visit in this case. The $\ell$-offset has an important property:

**Lemma 2.5** *The $\ell$-offset of a simple grid polygon, $P$, has at least $8\ell$ edges fewer than $P$.*

**Proof.** First, we can cut off blind alleys that are narrower than $2\ell$, because those parts of $P$ do not affect the $\ell$-offset. We walk clockwise around the boundary cells of the remaining polygon, see Figure 2.10. For every left turn the offset gains at most $2\ell$ edges and for every right turn the offset looses at least $2\ell$ edges. O'Rourke [150] showed that #vertices = 2·#reflex vertices+4

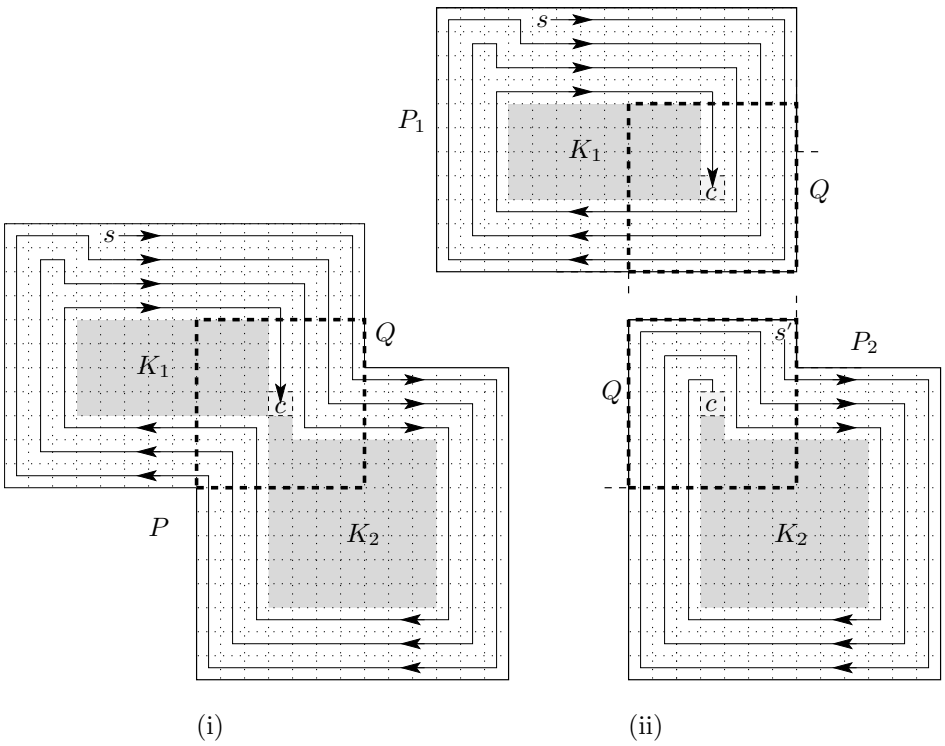



Figure 2.11: A decomposition of $P$ at the split cell $c$ and its handling in SmartDFS.

holds for orthogonal polygons, so there are four more right turns than left turns. □

Definition 2.4 allows us to specify the detection and handling of a split cell in SmartDFS. We start with the handling of a split cell and defer split cell detection.

Let us consider the situation shown in Figure 2.11(i) to explain the handling of a split cell. SmartDFS has just met the first split cell, $c$, in the fourth layer of $P$. $P$ divides into three parts:

$$P = K_1 \dot{\cup} K_2 \dot{\cup} \{\text{ visited cells of } P\},$$

where $K_1$ and $K_2$ denote the connected components of the set of unvisited cells. In this case it is reasonable to explore the component $K_2$ first, because the start cell $s$ is closer to $K_1$; that is, we can extend $K_1$ with $\ell$ layers, such that the resulting polygon contains the start cell $s$.

More generally, we want to divide our polygon $P$ into two parts, $P_1$ and $P_2$, so that each of them is an extension of the two components. Both polygons overlap in the area around the split cell $c$. At least one of these polygons contains the start cell. If only one of the polygons contains $s$, we want our strategy to explore this part at last, expecting that in this part the path from the last visited cell back to $s$ is the shorter than in the other

part. Vice versa, if there is a polygon that does *not* contain $s$, we explore the corresponding component first. In Figure 2.11, SmartDFS recursively enters $K_2$, returns to the split cell $c$, and explores the component $K_1$ next.

In the preceding example, there is only one split cell in $P$, but in general there will be a sequence of split cells, $c_1, \ldots, c_k$. In this case, we apply the handling of split cells in a recursive way; that is, if a split cell $c_{i+1}, 1 \leq i < k$, is detected in one of the two components occurring at $c_i$ we proceed the same way as described earlier. Only the role of the start cell is now played by the preceding split cell $c_i$. In the following, the term *start cell* always refers to the start cell of the current component; that is, either to $s$ or to the previously detected split cell. Further, it may occur that three components arise at a split cell, see Figure 2.14(i) on page 28. We handle this case as two successive polygon splits occurring at the same split cell.

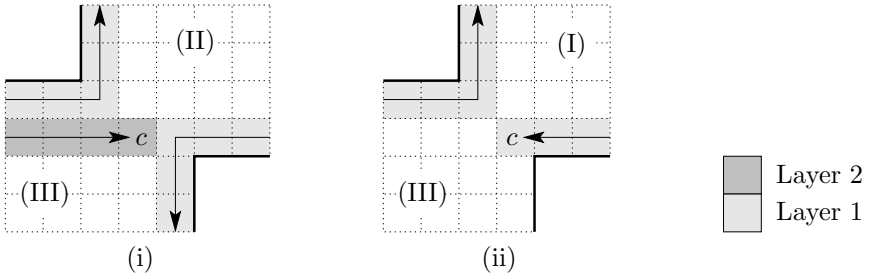



Figure 2.12: Several types of components.

**Visiting Order**

We use the layer numbers to decide which component we have to visit at last. Whenever a split cell occurs in layer $\ell$, every component is one of the following types, see Figure 2.12:

I. $K_i$ is *completely* surrounded by layer $\ell$[5]
II. $K_i$ is *not* surrounded by layer $\ell$
III. $K_i$ is *partially* surrounded by layer $\ell$

There are two cases, in which SmartDFS switches from a layer $\ell - 1$ to layer $\ell$. Either it reaches the first cell of layer $\ell - 1$ in the current component and thus passes the start cell—see, for example, the switch from layer 1 to layer 2 in Figure 2.13—, or it hits another cell of layer $\ell - 1$ but no polygon split occurs, such as the switch from layer 2 to layer 3 in in Figure 2.13. In the second case, the considered start cell must be located in a narrow passage that is completely explored; otherwise, the strategy would be able to reach the first cell of layer $\ell - 1$ as in the first case. In both cases the part of $P$ surrounding a component of type III contains the first cell of the

[5]More precisely, the part of layer $\ell$ that surrounds $K_i$ is completely visited. For convenience, we use the slightly sloppy, but shorter form.

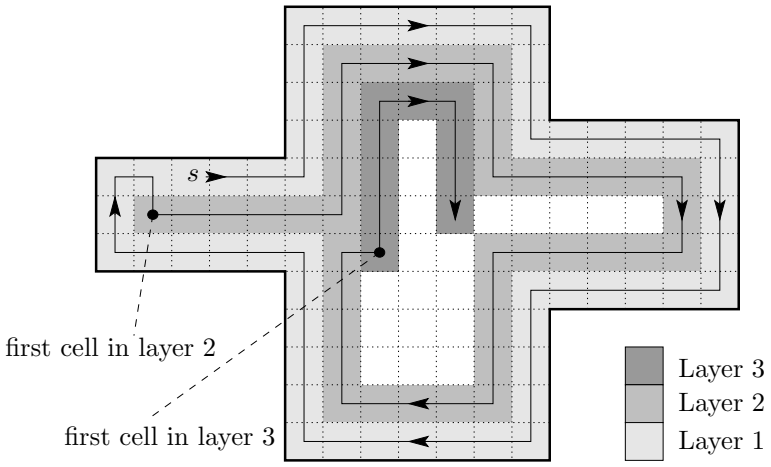



Figure 2.13: Switching the current layer.

current layer $\ell$ as well as the start cell. Therefore, it is reasonable to explore the component of type III at last.

There are two cases, in which no component of type III exists when a split cell is detected:

1. The part of the polygon that contains the preceding start cell is explored completely, see for example Figure 2.14(i). In this case the order of the components makes no difference.[6]

2. Both components are completely surrounded by a layer, because the polygon split and the switch from one layer to the next occurs within the same cell, see Figure 2.14(ii). A step that follows the left-hand rule will move towards the start cell, so we just omit this step. More precisely, if the the robot can walk to the left, we prefer a step forward to a step to the right. If the robot cannot walk to the left but straight forward, we proceed with a step to the right.

We proceed with the rule in case 2 whenever there is no component of type III, because the order in case 1 does not make a difference.

**An Upper Bound on the Number of Steps**

For the analysis of our strategy we consider two polygons, $P_1$ and $P_2$, as follows. Let $Q$ be the square of width $2q+1$ around $c$ with

$$q := \begin{cases} \ell, & \text{if } K_2 \text{ is of type I} \\ \ell - 1, & \text{if } K_2 \text{ is of type II} \end{cases},$$

[6] In Figure 2.14(i) we gain two steps, if we explore the part left to the splitcell at last and do not return to the split cell after this part is completely explored, but return immediately to the start cell. But decisions like this require facts of much more global type than we consider up to now. However, for the analysis of our strategy and the upper bound shortcuts like this do not matter.

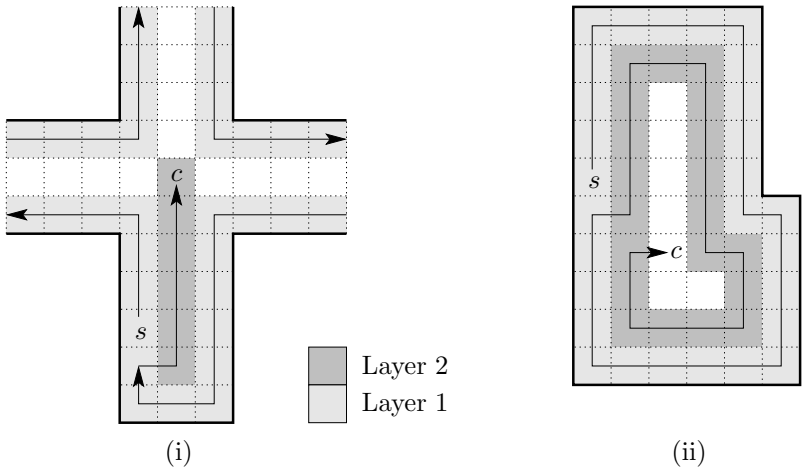



Figure 2.14: No component of type III exists.

where $K_2$ denotes the component that is explored first, and $\ell$ denotes the layer in which the split cell was found. We choose $P_2 \subset P \cup Q$ such that $K_2 \cup \{c\}$ is the $q$-offset of $P_2$, and $P_1 := ((P \backslash P_2) \cup Q) \cap P$, see Figure 2.11. The intersection with $P$ is necessary, because $Q$ may exceed the boundary of $P$. Note that at least $P_1$ contains the preceding start cell. There is an arbitrary number of polygons $P_2$, such that $K_2 \cup \{c\}$ is the $q$-offset of $P_2$, because blind alleys of $P_2$ that are not wider than $2q$ do not affect the $q$-offset. To ensure a unique choice of $P_1$ and $P_2$, we require that both $P_1$ and $P_2$ are connected, and both $P \cup Q = P_1 \cup P_2$ and $P_1 \cap P_2 \subseteq Q$ are satisfied.

The choice of $P_1, P_2$ and $Q$ ensures that the robot's path in $P_1 \backslash Q$ and in $P_2 \backslash Q$ do not change compared to the path in $P$. The parts of the robot's path that lead from $P_1$ to $P_2$ and from $P_2$ to $P_1$ are fully contained in the square $Q$. Just the parts inside $Q$ are bended to connect the appropriate paths inside $P_1$ and $P_2$, see Figure 2.11 and Figure 2.15.

In Figure 2.11, $K_1$ is of type III and $K_2$ is of type II. A component of type I occurs, if we detect a split cell as shown in Figure 2.15. Note that $Q$ may exceed $P$, but $P_1$ and $P_2$ are still well-defined.

Remark that we do not guarantee that the path from the last visited cell back to the corresponding start cell is the shortest possible path. See, for example, Figure 2.16: A split cell is met in layer 2. Following the preceding rule, SmartDFS enters $K_2$ first, returns to $c$, explores $K_1$, and returns to $s$. A path that visits $K_1$ first and moves from the upper cell in $K_2$ to $s$ is slightly shorter. A case like this may occur if the first cell of the current layer lies in $Q$. However, we guarantee that there is only one return path in $P_1 \backslash Q$ and in $P_2 \backslash Q$; that is, only one path leads from the last visited cell back to the preceding start cell causing double visits of cells.

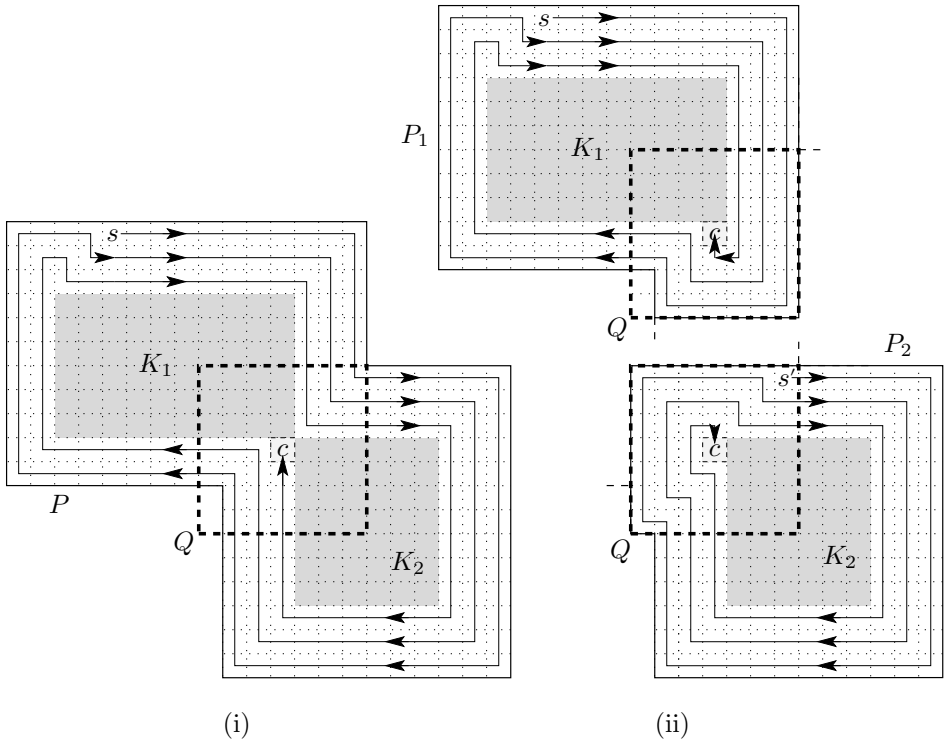



Figure 2.15: The component $K_2$ is of type I. The square $Q$ may exceed $P$.

We want to visit every cell in the polygon and to return to $s$. Every strategy needs at least $C(P)$ steps to fulfill this task, where $C(P)$ denotes the number of cells in $P$. Thus, we can split the overall length of the exploration path, $\pi$, into two parts, $C(P)$ and excess$(P)$, with $|\pi| = C(P) + \text{excess}(P)$. $C(P)$ is a lower bound on the number of steps that are needed for the exploration task, whereas excess$(P)$ is the number of additional cell visits.

Because SmartDFS recursively explores $K_2 \cup \{c\}$, we want to apply the upper bound inductively to the component $K_2 \cup \{c\}$. If we explore $P_1$ with SmartDFS until $c$ is met, the set of unvisited cells of $P_1$ is equal to $K_1$, because the path outside $Q$ do not change. Thus, we can apply our bound inductively to $P_1$, too. The following lemma gives us the relation between the path lengths in $P$ and the path lengths in the two components.

**Lemma 2.6** *Let $P$ be a simple grid polygon. Let the robot visit the first split cell, $c$, which splits the unvisited cells of $P$ into two components $K_1$ and $K_2$, where $K_2$ is of type I or II. With the preceding notations we have*

$$\text{excess}(P) \leq \text{excess}(P_1) + \text{excess}(K_2 \cup \{c\}) + 1 \ .$$

**Proof.** The strategy SmartDFS has reached the split cell $c$ and explores $K_2 \cup \{c\}$ with start cell $c$ first. Because $c$ is the first split cell, there is

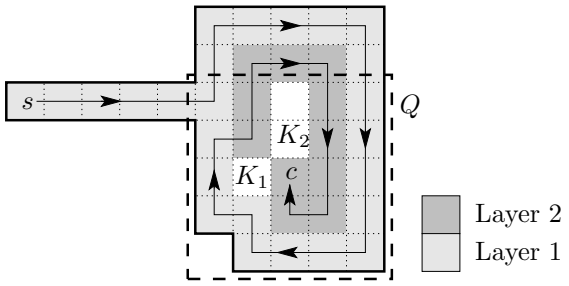



Figure 2.16: The order of components is not necessarily optimal.

no excess in $P_2 \backslash (K_2 \cup \{c\})$ and it suffices to consider excess$(K_2 \cup \{c\})$ for this part of the polygon. After $K_2 \cup \{c\}$ is finished, the robot returns to $c$ and explores $K_1$. For this part we take excess$(P_1)$ into account. Finally, we add one single step, because the split cell $c$ is visited twice: once, when SmartDFS detects the split and once more after the exploration of $K_2 \cup \{c\}$ is finished. Altogether, the given bound is achieved. □

$c$ is the first split cell in $P$, so $K_2 \cup \{c\}$ is the $q$-offset of $P_2$ and we can apply Lemma 2.5 to bound the number of boundary edges of $K_2 \cup \{c\}$ by the number of boundary edges of $P_2$. The following lemma allows us to charge the number of edges in $P_1$ and $P_2$ against the number of edges in $P$ and $Q$.

**Lemma 2.7** *Let $P$ be a simple grid polygon, and let $P_1, P_2$ and $Q$ be defined as earlier. The number of edges satisfy the equation*

$$E(P_1) + E(P_2) = E(P) + E(Q) \; .$$

**Proof.** Obviously, two arbitrary polygons $P_1$ and $P_2$ always satisfy

$$E(P_1) + E(P_2) = E(P_1 \cup P_2) + E(P_1 \cap P_2) \; .$$

Let $Q' := P_1 \cap P_2$. Note that $Q'$ is not necessarily the same as $Q$, see, for example, Figure 2.15. With $P_1 \cup P_2 = P \cup Q$ we have

$$\begin{aligned} E(P_1) + E(P_2) &= E(P_1 \cap P_2) + E(P_1 \cup P_2) \\ &= E(Q') + E(P \cup Q) \\ &= E(Q') + E(P) + E(Q) - E(P \cap Q) \\ &= E(P) + E(Q) \end{aligned}$$

The latter equation holds because $Q' = P \cap Q$. □

Finally, we need an upper bound for the length of a path inside a grid polygon.

**Lemma 2.8** *Let $\pi$ be the shortest path between two cells in a grid polygon $P$. The length of $\pi$ is bounded by*

$$|\pi| \leq \frac{1}{2}E(P) - 2 \,.$$

**Proof.** W. l. o. g. we can assume that the start cell, $s$, and the target cell, $t$, of $\pi$ belong to the first layer of $P$, because we are searching for an upper bound for the shortest path between two arbitrary cells.

Observe the path $\pi_L$ from $s$ to $t$ in the first layer that follows the boundary of $P$ clockwise and the path $\pi_R$ that follows the boundary counterclockwise. The number of edges along these paths is at least four greater than the number of cells visited by $\pi_L$ and $\pi_R$ using an argument similar to the proof of Lemma 2.5. Therefore we have:

$$|\pi_L| + |\pi_R| \leq E(P) - 4.$$

In the worst case, both paths have the same length, so $|\pi(s,t)| = |\pi_L| = |\pi_R|$ holds. With this we have

$$2 \cdot |\pi(s,t)| \leq E(P) - 4 \implies |\pi(s,t)| \leq \frac{1}{2}E(P) - 2.$$

□

Now, we are able to show our main theorem:

**Theorem 2.9** *Let $P$ be a simple grid polygon with $C$ cells and $E$ edges. $P$ can be explored with*

$$S \leq C + \frac{1}{2}E - 3$$

*steps. This bound is tight.*

**Proof.** $C$ is the number of cells and thus a lower bound on the number of steps that are needed to explore the polygon $P$. We show by an induction on the number of components that $\text{excess}(P) \leq \frac{1}{2}E(P) - 3$ holds.

For the induction base we consider a polygon without any split cell: SmartDFS visits each cell and returns on the shortest path to the start cell. Because there is no polygon split, all cells of $P$ can be visited by a path of length $C - 1$. By Lemma 2.8 the shortest path back to the start cell is not longer than $\frac{1}{2}E - 2$; thus, $\text{excess}(P) \leq \frac{1}{2}E(P) - 3$ holds.

Now, we assume that there is more than one component during the application of SmartDFS. Let $c$ be the first split cell detected in $P$. When SmartDFS reaches $c$, two new components, $K_1$ and $K_2$, occur. We consider the two polygons $P_1$ and $P_2$ defined as earlier, using the square $Q$ around $c$.

W. l. o. g. we assume that $K_2$ is recursively explored first with $c$ as start cell. After $K_2$ is completely explored, SmartDFS proceeds with the remaining polygon. As shown in Lemma 2.6 we have

$$\text{excess}(P) \leq \text{excess}(P_1) + \text{excess}(K_2 \cup \{c\}) + 1 \; .$$

Now, we apply the induction hypothesis to $P_1$ and $K_2 \cup \{c\}$ and get

$$\text{excess}(P) \leq \frac{1}{2}E(P_1) - 3 + \frac{1}{2}E(K_2 \cup \{c\}) - 3 + 1 \; .$$

By applying Lemma 2.5 to the $q$-offset $K_2 \cup \{c\}$ of $P_2$ we achieve

$$\begin{aligned} \text{excess}(P) &\leq \frac{1}{2}E(P_1) - 3 + \frac{1}{2}(E(P_2) - 8q) - 3 + 1 \\ &= \frac{1}{2}(E(P_1) + E(P_2)) - 4q - 5 \; . \end{aligned}$$

From Lemma 2.7 we conclude $E(P_1) + E(P_2) \leq E(P) + 4(2q+1)$. Thus, we get $\text{excess}(P) \leq \frac{1}{2}E(P) - 3$.

In Section 2.1 we have already seen that the bound is exactly achieved in polygons that do not contain any $2 \times 2$-square of free cells. □

**Competitive Factor**

So far we have shown an upper bound on the number of steps needed to explore a polygon that depends on the number of cells and edges in the polygon. Now, we want to analyze SmartDFS in the competitive framework.

Corridors of width 1 or 2 play a crucial role in the following, so we refer to them as *narrow passages*. More precisely, a cell, $c$, belongs to a narrow passage, if $c$ can be removed without changing the layer number of any other cell.

It is easy to see that narrow passages are explored optimally: In corridors of width 1 both SmartDFS and the optimal strategy visit every cell twice, and in the other case both strategies visit every cell exactly once.

We need two lemmata to show a competitive factor for SmartDFS. The first one gives us a relation between the number of cells and the number of edges for a special class of polygons.

**Lemma 2.10** *For a simple grid polygon, $P$, with $C(P)$ cells and $E(P)$ edges, and without any narrow passage or split cells in the first layer, we have*

$$E(P) \leq \frac{2}{3}\,C(P) + 6 \, .$$

**Proof.** Consider a simple polygon, $P$. We successively remove a row or column of at least three boundary cells, maintaining our assumption that the polygon has no narrow passages or split cells in the first layer. These

assumptions ensure that we can always find such a row or column. Thus, we remove at least three cells and at most two edges. This decomposition ends with a $3 \times 3$ block of cells that fulfills $E = \frac{2}{3}C + 6$. Now, we reverse our decomposition; that is, we successively add all rows and columns until we end up with $P$. In every step, we add at least three cells and at most two edges. Thus, $E \leq \frac{2}{3}C + 6$ is fulfilled in every step. □

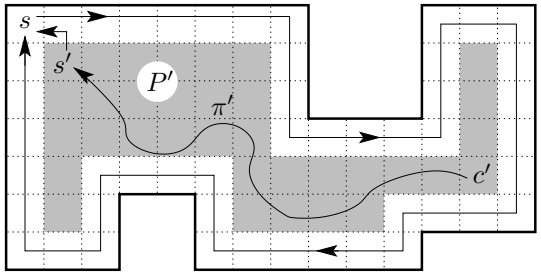



Figure 2.17: For polygons without narrow passages or split cells in the first layer, the last explored cell, $c'$, lies in the 1-offset, $P'$ (shaded).

For the same class of polygons, we can show that SmartDFS behaves slightly better than the bound in Theorem 2.9.

**Lemma 2.11** *A simple grid polygon, $P$, with $C(P)$ cells and $E(P)$ edges, and without any narrow passage or split cells in the first layer can be explored using no more steps than*

$$S(P) \leq C(P) + \frac{1}{2}E(P) - 5\,.$$

**Proof.** In Theorem 2.9 we have seen that $S(P) \leq C(P) + \frac{1}{2}E(P) - 3$ holds. To show this theorem, we used Lemma 2.8 on page 31 as an upper bound for the shortest path back from the last explored cell to the start cell. Lemma 2.8 bounds the shortest path from a cell, $c$, in the first layer of $P$ to the cell $c'$ that maximizes the distance to $c$ inside $P$; thus, $c'$ is located in the first layer of $P$, too.

Because $P$ has neither narrow passages nor split cells in the first layer, we can explore the first layer of $P$ completely before we visit another layer, see Figure 2.17. Therefore, the last explored cell, $c'$, of $P$ is located in the 1-offset of $P$. Let $P'$ denote the 1-offset of $P$, and $s'$ the first visited cell in $P'$. Remark that $s$ and $s'$ are at least touching each other, so the length of a shortest path from $s'$ to $s$ is at most 2. Now, the shortest path, $\pi$, from $c'$ to $s$ in $P$ is bounded by a shortest path, $\pi'$, from $c'$ to $s'$ in $P'$ and a shortest path from $s'$ to $s$:

$$|\pi| \leq |\pi'| + 2\,.$$

The path $\pi'$, in turn, is bounded using Lemma 2.8 by

$$|\pi'| \leq E(P') - 2\,.$$

By Lemma 2.5 (page 24), $E(P') \leq E(P) - 4$ holds, and altogether we get

$$|\pi| \leq E(P) - 4\,,$$

which is two steps shorter than stated in Lemma 2.8. □

Now, we can prove the following

**Theorem 2.12** *The strategy SmartDFS is $\frac{4}{3}$-competitive.*

**Proof.** Let $P$ be a simple grid polygon. In the first stage, we remove all narrow passages from $P$ and get a sequence of (sub-)polygons $P_i$, $i = 1, \ldots, k$, without narrow passages. For every $P_i$, $i = 1, \ldots, k-1$, the optimal strategy in $P$ explores the part of $P$ that corresponds to $P_i$ up to the narrow passage that connects $P_i$ with $P_{i+1}$, enters $P_{i+1}$, and fully explores every $P_j$ with $j \geq i$. Then it returns to $P_i$ and continues with the exploration of $P_i$. Further, we already know that narrow passages are explored optimally. This allows us to consider every $P_i$ separately without changing the competitive factor of $P$.

Now, we observe a (sub-)polygon $P_i$. We show by induction on the number of split cells in the first layer that $S(P_i) \leq \frac{4}{3}C(P_i) - 2$ holds. Note that this is exactly achieved in polygons of size $3 \times m$, $m$ even, see Figure 2.18.


SmartDFS optimal strategy

Figure 2.18: In a corridor of width 3 and even length, $S(P) = \frac{4}{3}\,S_{\mathrm{Opt}}(P) - 2$ holds.

If $P_i$ has no split cell in the first layer (induction base), we can apply Lemma 2.11 and Lemma 2.10:

$$\begin{aligned} S(P_i) &\leq C(P_i) + \frac{1}{2}\,E(P_i) - 5 \\ &\leq C(P_i) + \frac{1}{2}\left(\frac{2}{3}\,C(P_i) + 6\right) - 5 \\ &= \frac{4}{3}\,C(P_i) - 2\,. \end{aligned}$$

Two cases occur if we meet a split cell, $c$, in the first layer, see Figure 2.19. In the first case, the new component was never visited before (component of type II, see page 26). Here, we define $Q := \{c\}$. The second case occurs, because the robot meets a cell, $c'$, that is in the first layer and touches the current cell, $c$, see for example Figure 2.19(ii) and (iii). Let $Q$ be the smallest rectangle that contains both $c$ and $c'$.

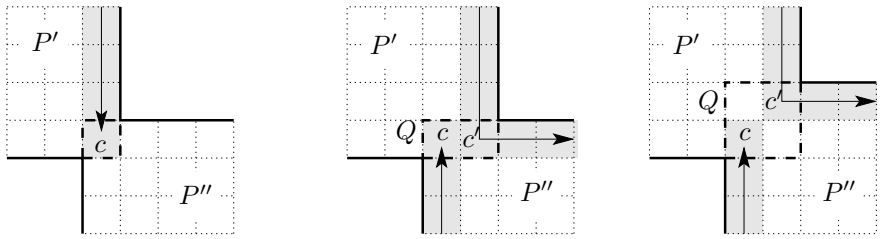



Figure 2.19: Three cases of split cells, (i) component of type II, (ii) and (iii) component of type I.

Similar to the proof of Theorem 2.9, we split the polygon $P_i$ into two parts, both including $Q$. Let $P''$ denote the part that includes the component of type II or III, $P'$ the other part. For $|Q| = 1$, see Figure 2.19(i), we conclude $S(P_i) = S(P') + S(P'')$ and $C(P_i) = C(P') + C(P'') - 1$. Applying the induction hypothesis to $P'$ and $P''$ yields

$$\begin{aligned} S(P_i) &= S(P') + S(P'') \\ &\leq \frac{4}{3}C(P') - 2 + \frac{4}{3}C(P'') - 2 \\ &= \frac{4}{3}C(P_i) + \frac{4}{3} - 4 \quad < \quad \frac{4}{3}C(P_i) - 2\,. \end{aligned}$$

For $|Q| \in \{2, 4\}$ we gain some steps by merging the polygons. If we consider $P'$ and $P''$ separately, we count the steps from $c'$ to $c$—or vice versa—in both polygons, but in $P_i$ the path from $c'$ to $c$ is replaced by the exploration path in $P''$. Thus, we have $S(P_i) = S(P') + S(P'') - |Q|$ and $C(P_i) = C(P') + C(P'') - |Q|$. This yields

$$\begin{aligned} S(P_i) &= S(P') + S(P'') - |Q| \\ &\leq \frac{4}{3}C(P') - 2 + \frac{4}{3}C(P'') - 2 - |Q| \\ &= \frac{4}{3}C(P_i) + \frac{1}{3}(|Q| - 6) - 2 \quad < \quad \frac{4}{3}C(P_i) - 2\,. \end{aligned}$$

The optimal strategy needs at least $C$ steps, which, altogether, yields a competitive factor of $\frac{4}{3}$. □

**Split Cell Detection**

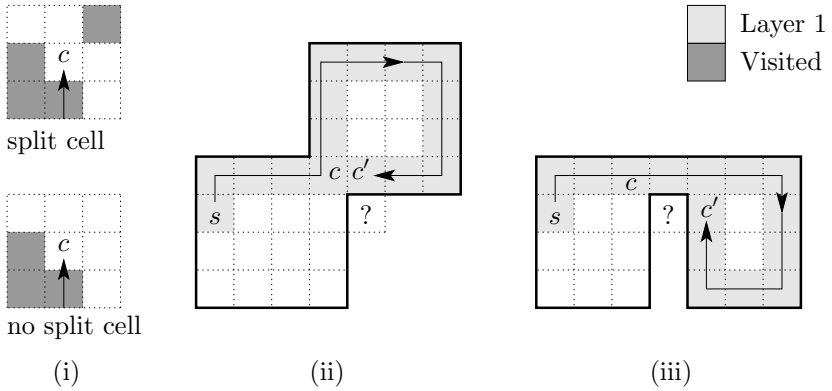



Figure 2.20: (i) Detecting a split cell, (ii) and (iii) a polygon split occurs in layer 1.

Finally, we describe the *detection* of a split cell. In the first instance, let us assume that the robot already moves in a layer $\ell > 1$. In Section 2.2.1 we defined that a split cell divides the graph of unexplored cells into two parts when the split cell is visited. Because the polygon is simple, we can determine a global split using a local criterion. We observe the eight cells surrounding the current robot's position. If there is more than one connected set of visited cells in this block, the current robot position is obviously a split cell, see Figure 2.20(i). Remark that we can execute this test although the robot's sensors do not allow us to access all eight cells around the current position. We are interested only in visited cells, so we can use the robot's map of visited cells for our test in layers $\ell > 1$. Unfortunately, this test method fails in layer 1, because the robot does not know the "layer 0", the polygon walls. However, we want to visit the component that has no visited cell in the current layer (type II) first; therefore, a step that follows the left-hand rule is correct. The strategy behaves correctly, although we do not report the splitcell explicitly. See, for example, Figure 2.20(ii) and (iii): In both cases the polygon split cannot be detected in $c$, because the cell marked with '?' is not known at this time. The split will be identified and handled correctly in $c'$.

## 2.3 Exploring Polygons with Holes

In an environment with obstacles (*holes*) it is not obvious how to detect and to handle split cells. When a polygon split is detected, the robot may be far away from the split cell, because it had no chance to recognize the split before reaching its current position. For example, in Figure 2.21(i) the robot has surrounded one single obstacle and $c$ is a split cell, whereas in (ii) there are two obstacles and $c$ is no split cell. Both situations cannot be distinguished until the cell $c'$ is reached. So we use a different strategy to explore environments with obstacles.

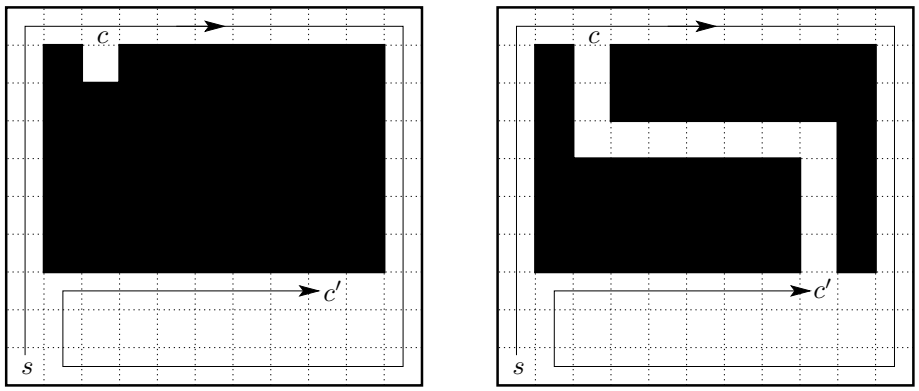



Figure 2.21: In an environment with obstacles, the robot may detect a split on a position far away from the splitcell, (i) $c$ was a split cell, (ii) $c$ was no split cell.

### 2.3.1 An Exploration Strategy

The basic idea of our strategy, CellExplore, is to *reserve* all cells right to the covered path for the way back. As in SmartDFS we use the left-hand rule; that is, the robot proceeds keeping the polygon's boundary or the reserved cells on its left side. CellExplore uses two modes. In the forward mode the robot enters undiscovered parts of the polygon, and in the backward mode the robot leaves known parts, see Algorithm 2.3 on page 39. We require that the robot starts with its back to a wall.

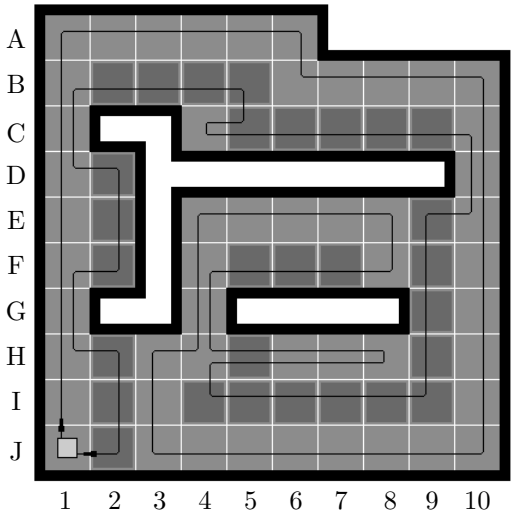



Figure 2.22: Example of an exploration tour produced by CellExplore (Screenshot using [77]; the white cells are holes, dark gray cells are reserved).

Figure 2.22 shows an example of an exploration tour. The robot starts in J1 and explores the polygon in the forward mode until F8 is reached. There, the robot switches to the backward mode and explores the reserved cells F7–F5. The path from F5 to H5 is blocked by the hole in G5, so the robot walks on the cells F4–H4 which have been visited already in the forward mode. In H5 the robot discovers the unreserved and unexplored cell H6, switches back to the forward mode and explores the cells H6–H8. Note that no cells can be reserved in this case, because the cells I6–I8 have already been reserved during the exploration of J8–J6. Therefore, the robot walks the same path back to H5 and continues the return path in the backward mode. In the forward mode, the robot could not reserve a cell from H3, so we move via H4 to I4 and proceed with the return path in the backward mode. The cells D9, C2 and G2 are blocked, so the robot has to circumvent these cells using visited cells. In C5 another unreserved and unexplored cell is discovered, so we switch to the forward mode and visit C4.

**Algorithm 2.3** CellExplore

**Forward mode:**

- The polygon is explored following the left-hand rule: For every entered cell the robot tries to extend its path to an adjacent, unexplored, and unreserved cell, preferring a step to the left[7] over a straight step over a step to the right.
- All unexplored and unreserved cells right to the covered path are reserved for the return path by pushing them onto a stack.[8] If no cell right to the robot's current position can be reserved—because there is a hole or the corresponding cell is already reserved or explored—the robot's position is pushed onto the stack for the return path.
- Whenever no step in the forward mode is possible, the strategy enters the backward mode.

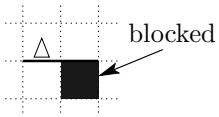
**Backward mode:**

- The robot walks back along the reserved return path.
- Whenever an unexplored and unreserved cell appears adjacent to the current position, the forward mode is entered again.

[7] A "step to the left" or "turn left" means that the robot turns 90° counterclockwise and moves one cell forward. Analogously with "step to the right" or "turn right".

[8] If the robot turns left, we reserve three cells: right hand, straight forward, and forward-right to the robot's position. Note that we store the markers only in the robot's memory. This allows us to reserve cells that only touch the current cell, even if we are not able to determine whether these cells are free or blocked.

### 2.3.2 The Analysis of CellExplore




We analyze CellExplore in three steps: First, we analyze the single steps of the strategy with a local view and with one assumption concerning the robot's initial position, see the figure on the right. This results in a bound that depends on the number of left turns that are needed to explore the polygon. Then we discard the assumption, and finally we consider some global arguments to replace the number of left turns with parameters of the polygon.
To analyze our strategy CellExplore we use the following observations:

- CellExplore introduces a dissection of all cells into cells that are explored in the forward mode, denoted by $\mathcal{F}$, and cells that are explored in the backward mode, $\mathcal{B}$: $\mathcal{C} = \mathcal{F} \dot{\cup} \mathcal{B}$.
- All cells that are explored in the forward mode can be uniquely classified by the way the robot leaves them: either it makes a step to the left, a step forward or a step to the right. $\mathcal{F} = \mathcal{F}_L \dot{\cup} \mathcal{F}_F \dot{\cup} \mathcal{F}_R$.
- CellExplore defines a mapping $\varphi : \mathcal{B} \longrightarrow \mathcal{F}$, which assigns to every cell $c \in \mathcal{B}$ one cell $d \in \mathcal{F}$, so that $c$ was reserved while $d$ was visited.
- There exists a subset $\mathcal{D} \subseteq \mathcal{B}$ of the cells that have an unexplored and unreserved neighbor and the strategy switches from backward mode to forward mode. We call these cells *division cells*.

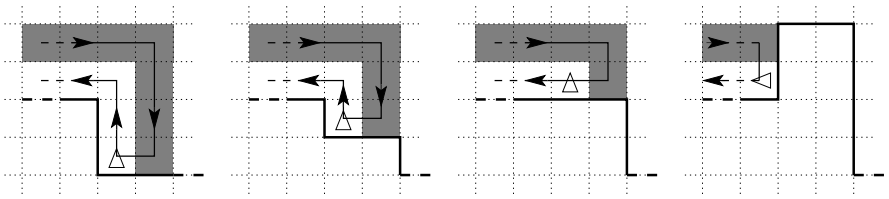



Figure 2.23: Decomposing a polygon. The shaded part shows the reserved cells.

We will analyze CellExplore by an induction over the cells in $\mathcal{F}$. Starting with the given polygon, $P$, and the given start cell, $s$, we can define a sequence $P_{k,i}$ of polygons ($P_{0,0} := P$) with start cells $s_{k,i}$ as follows: $P_{k,i+1}$ arises from $P_{k,i}$ by removing the start cell $s_{k,i}$ and all cells that are reserved in the first step in $P_{k,i}$ (i. e., every cell $c$ with $\varphi(c) = s_{k,i}$). The start cell $s_{k,i+1}$ in the new polygon $P_{k,i+1}$ is the cell that the robot enters with its first step in $P_{k,i}$, see Figure 2.23. The reserved cells in this and all following figures are shown shaded; $\triangle$ denotes the start cell.

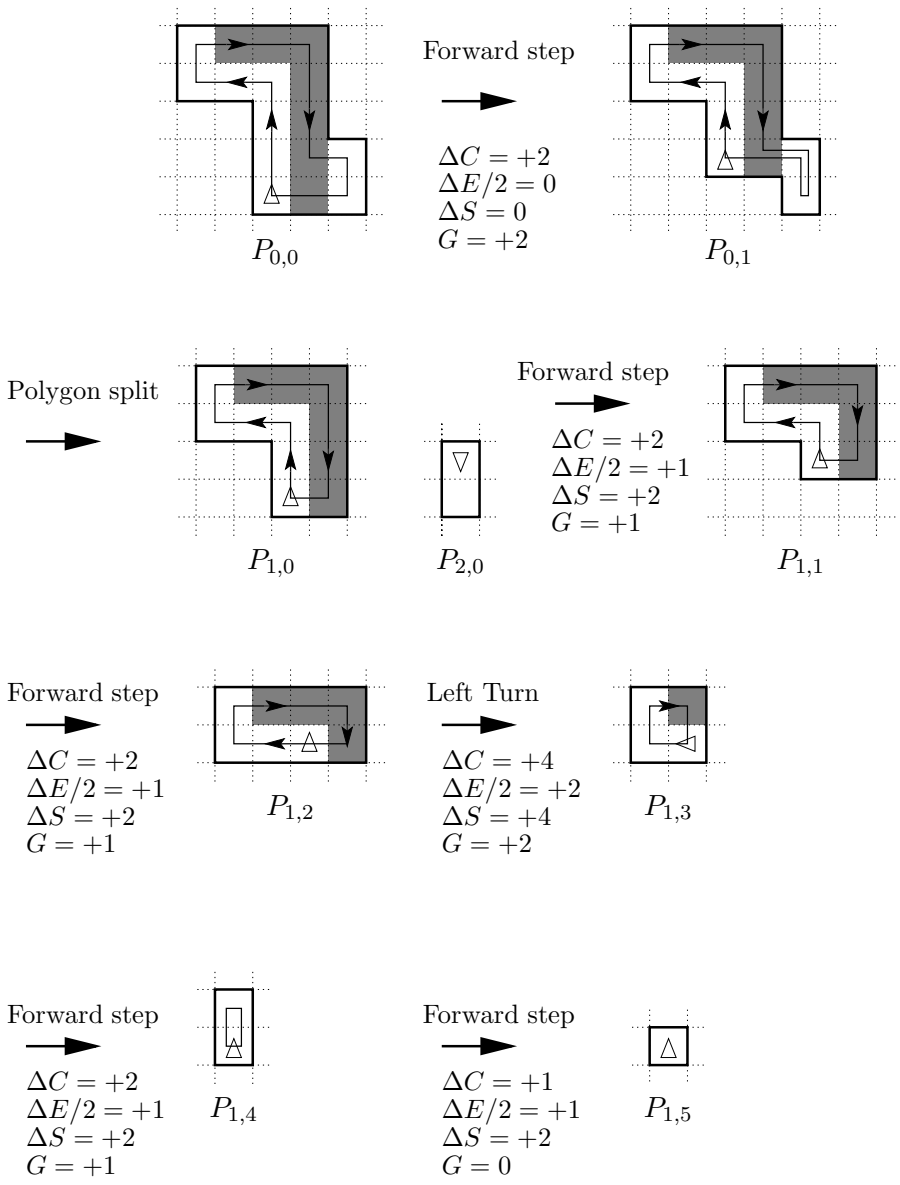



Figure 2.24: Decomposing a polygon. △ denotes the start cell and the initial direction. $\Delta C, \Delta E$ and $\Delta S$ denote the differences in the number of cells, edges, and steps, respectively. $G$ denotes the balance.

There is nothing to consider when the strategy enters the backward mode, because we remove all cells that are explored in the backward mode together with the forward cells. But what happens, if a division cell occurs; that is, the strategy switches from the backward mode to the forward mode?

**Lemma 2.13** *If one of the cells reserved in the first step in $P_{k,i}$ is a division cell, then $P_{k,i}$ is split by removing $s_{k,i}$ and all cells that are reserved in this step (i. e., all cells $c$ with $\varphi(c) = s_{k,i}$) into two or more not-connected components.*

**Proof.** Consider two components, $P_1$ and $P_2$, that are connected in $P$ by some cells $\mathcal{X} \subset \mathcal{B}$. Let $c_j$ be the first cell in $\mathcal{X}$ that is discovered on the return path, thus, $P_2$ is entered via $c_j$. In our successive decomposition, $\varphi(c_j)$ is the start cell of a polygon $P_{k,i}$. In $P_{k,i}$ all cells explored before $\varphi(c_j)$ are already removed. If there would be another connection, $c_\ell$, between $P_1$ and $P_2$ at this time, $c_\ell$ would be discovered before $c_j$ on the return path and $P_2$ would have been entered via $c_\ell$ in contradiction to our assumption that $P_2$ is entered via $c_j$. Thus, $c_j$ must be the last cell that connects $P_1$ and $P_2$. □

If one or more of the cells to be removed are division cells, the polygon $P_{k,i}$ is divided into subpolygons $P_{k+1,0}, P_{k+2,0}, \ldots$ that are analyzed separately, see Figure 2.25. See Figure 2.24 for a more comprehensive example for the successive decomposition of a polygon.

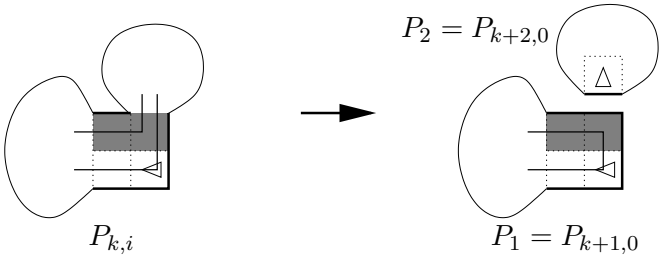



Figure 2.25: Handling of division cells.

Now, we are able give a first bound for the number of steps that CellExplore uses to explore a polygon.

**Lemma 2.14** *Let us assume that the cell behind and right hand to the robot's position*[9] *is blocked. The number of steps, $S$, used to explore a polygon with $C$ cells, $E$ edges and $H$ holes, is bounded by*

$$S \leq C + \frac{1}{2}E + H + 2L - 3,$$

[9] In other words, the cell southeast to the robot's current position if the current direction is *north*.

*where $L$ denotes the number of the robot's left turns.*

**Proof.** We observe the differences in the number of steps, cells, edges and holes between $P_{k,i}$ and $P_{k,i+1}$, and assume by induction that our upper bound for the length of the exploration tour holds for $P_{k,i+1}$ and for the separated subpolygons $P_{k+j,0}$. Therefore, we have to show that the limit is still not exceeded if we add the removed cells and merge the subpolygons. We want to show that the following inequation is satisfied in every step:

$$S \leq\ C + \frac{1}{2}E + H + 2L - 3.$$

Let $G$ denote the "profit" made by CellExplore; that is, the difference between the actual number of steps and the upper bound. With $G$, the preceding inequation is equivalent to

$$C + \frac{1}{2}E + H + 2L - G - S - 3 = 0.$$

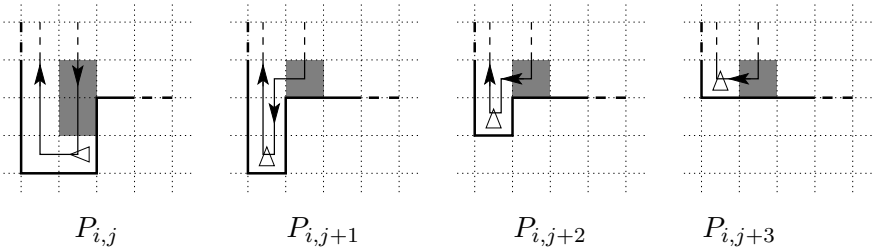



Figure 2.26: Decomposing a step to the right into several forward steps.

We have to consider three main cases: the division cells, the cells contained in $\mathcal{F}_L$ (left turns), and those contained in $\mathcal{F}_F \cup \mathcal{F}_R$ (forward steps and right turns). There is no need to consider right turns explicitly, because steps to the right can be handled as a sequence of forward steps, see Figure 2.26. The successive decomposition ends with one single cell ($C = 1, E = 4, H = 0$) for which $S = 0 = C + \frac{1}{2}E + H + 2L - 3$ holds (Induction base).

**Division cells**

If one of the cells that are reserved in the first step in $P_{k,i}$ is a division cell, $P_{k,i}$ is split into two polygons $P_{k+1,0}$ ($P_1$ for short) and $P_{k+2,0}$ ($P_2$ for short), see Lemma 2.13. We assume by induction that our upper bound is achieved in both polygons:

$$C_i + \frac{1}{2}E_i + H_i + 2L_i - G_i - S_i - 3 = 0, \quad i \in \{1, 2\}.$$

For the merge of $P_1$ and $P_2$ into one polygon $P$, we can state the following:

$$\begin{aligned} S &= S_1 + S_2 + \Delta S \\ E &= E_1 + E_2 + \Delta E \\ C &= C_1 + C_2 \\ H &= H_1 + H_2 \\ L &= L_1 + L_2 \\ G &= G_1 + G_2 + G_S, \end{aligned}$$

where $G_S$ denotes the profit made by merging the polygons.

We want to show that our bound is achieved in $P$:

$$\begin{aligned} & C + \frac{1}{2}E + H + 2L - G - S - 3 \\ &= C_1 + C_2 + \frac{1}{2}(E_1 + E_2 + \Delta E) + H_1 + H_2 + 2L_1 \\ &\quad + 2L_2 - G_1 - G_2 - G_S - S_1 - S_2 - \Delta S - 3 \\ &= \underbrace{C_1 + \frac{1}{2}E_1 + H_1 + 2L_1 - G_1 - S_1 - 3}_{=0} \\ &\quad + \underbrace{C_2 + \frac{1}{2}E_2 + H_2 + 2L_2 - G_2 - S_2 - 3}_{=0} \\ &\quad + \frac{1}{2}\Delta E - G_S - \Delta S + 3 \\ &= 0 \\ \iff G_S &= \frac{1}{2}\Delta E - \Delta S + 3 \end{aligned}$$

Thus, for every polygon split we have to observe $\Delta E$ and $\Delta S$. If $G_S$ is positive, we gain some steps by merging the polygons, if $G_S$ is negative, the merge incurs some costs.

The different configurations for a polygon split can be assigned to several classes, depending on the number of common edges between $P_1$ and $P_2$ and the way, the robot returns from $P_2$ to $P_1$—more precisely the distance between the step from $P_1$ to $P_2$ and the step from $P_2$ to $P_1$, compare for example Figure 2.29(3a) and (3b). Figure 2.29(i) and (ii) show some instances of one class. Because the actual values for $\Delta E$ and $\Delta S$ depend on only these two parameters, we do not list every possible polygon split in Figure 2.29, but some instances of every class.

The balances of the polygon splits are shown in table 2.1. Two cases have a negative balance, so we need one more argument to show that

these cases do not incur any costs. Observe that after splitting $P_{k,i}$ the polygon $P_1$ starts with a cell from $\mathcal{F}_L$ in the cases (1b)–(5). Removing this block of 4 cells, we gain +2, see the first line of Figure 2.31. This covers the costs for the polygon split in the cases (4a) and (5a).

**Forward steps**

We have to consider several cases of forward steps, see Figure 2.30. The table lists the differences in the number of steps ($\Delta S$), cells ($\Delta C$), edges ($\Delta E$) and holes ($\Delta H$) if we *add* the considered cells to $P_{k,i+1}$. The last column shows $G = \Delta C + \frac{1}{2}\Delta E + \Delta H - \Delta S$. After removing the observed block of cells, the remaining polygon must still be connected; otherwise, we would have to consider a polygon split first. Thus, there are some cases, in which $\Delta H$ must be greater than zero.

It turns out that all cases have a positive balance, except those that violate the assumption in Lemma 2.14 that the cell behind and right hand to the robot's position is blocked, see the cases marked with (*) in Figure 2.30. Notice that the configurations shown in Figure 2.27 are left turns instead of forward steps!

To show that we have analyzed all possible cell configurations for a step forward, we use the following observations. When the robot makes a step forward, we know the following: Both behind and left hand to the robot are walls (otherwise it would have turned left), in front of the robot is no wall (otherwise it could not make a step forward). Right hand to the robot may or may not be a wall. In the latter case, we have three edges of interest that may or may not be walls, yielding $2^3 = 8$ cases, which can be easily enumerated. Two special cases occur by taking into consideration that the robot may enter the observed block of cells from the upper cell or from the right cell, see for example Figure 2.30(5a) and (5b).

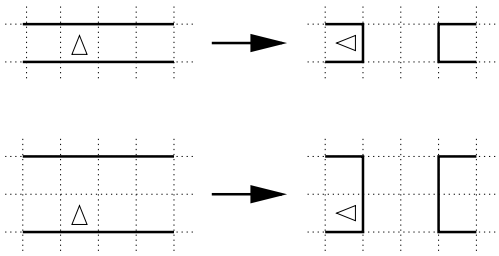
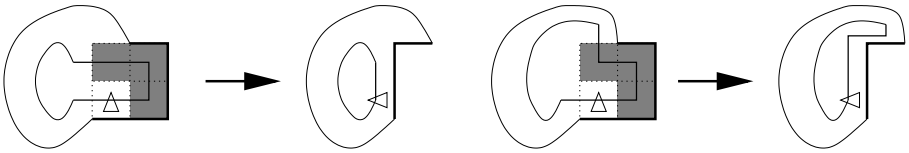


Figure 2.27: Configurations that are steps to the left instead of forward steps.

**Steps to the left**

Possible cases of left turns are shown in the figures 2.31–2.34. As in the previous case, the last column shows the balance. Again, we observe that all cases with a negative balance of $-3$ are a violation

of our assumption that the cell behind and right hand to the robot's position is blocked. The negative balances of $-2$ are compensated by the addend $2L$ in our bound.



(i) Block is entered from left
$\Delta S = 4$

(ii) Block is entered from upper cell
$\Delta S = 2$

Figure 2.28: Another class of left turns.

When counting the number of steps, we assumed that the robot enters the block of cells from the same direction as it left the block (from the left as shown in the figures). The robot may enter the block from the upper cell as shown in Figure 2.28, but this would only increase the balance.

The completeness of the cases can be shown with the same argument as in the previous case: We know that there is a wall behind the robot and left hand to the robot is no wall. Examining a block of four cells for a left turn, we have six edges that may or may not be walls, and we have three cells that may or may not be holes, yielding corresponding configurations of edges. □

| | $\Delta E$ | $\Delta S$ | $G_S$ |
|---|---|---|---|
| (1) | -2 | 2 | 0 |
| (2) | -4 | 0 | 1 |
| (3a) | -6 | 0 | 0 |
| (3b) | -6 | -2 | 2 |
| (4a) | -8 | 0 | -1 |
| (4b) | -8 | -2 | 1 |
| (4c) | -8 | -4 | 3 |
| (5a) | -10 | 0 | -2 |
| (5b) | -10 | -2 | 0 |

Table 2.1: Balances of polygon splits

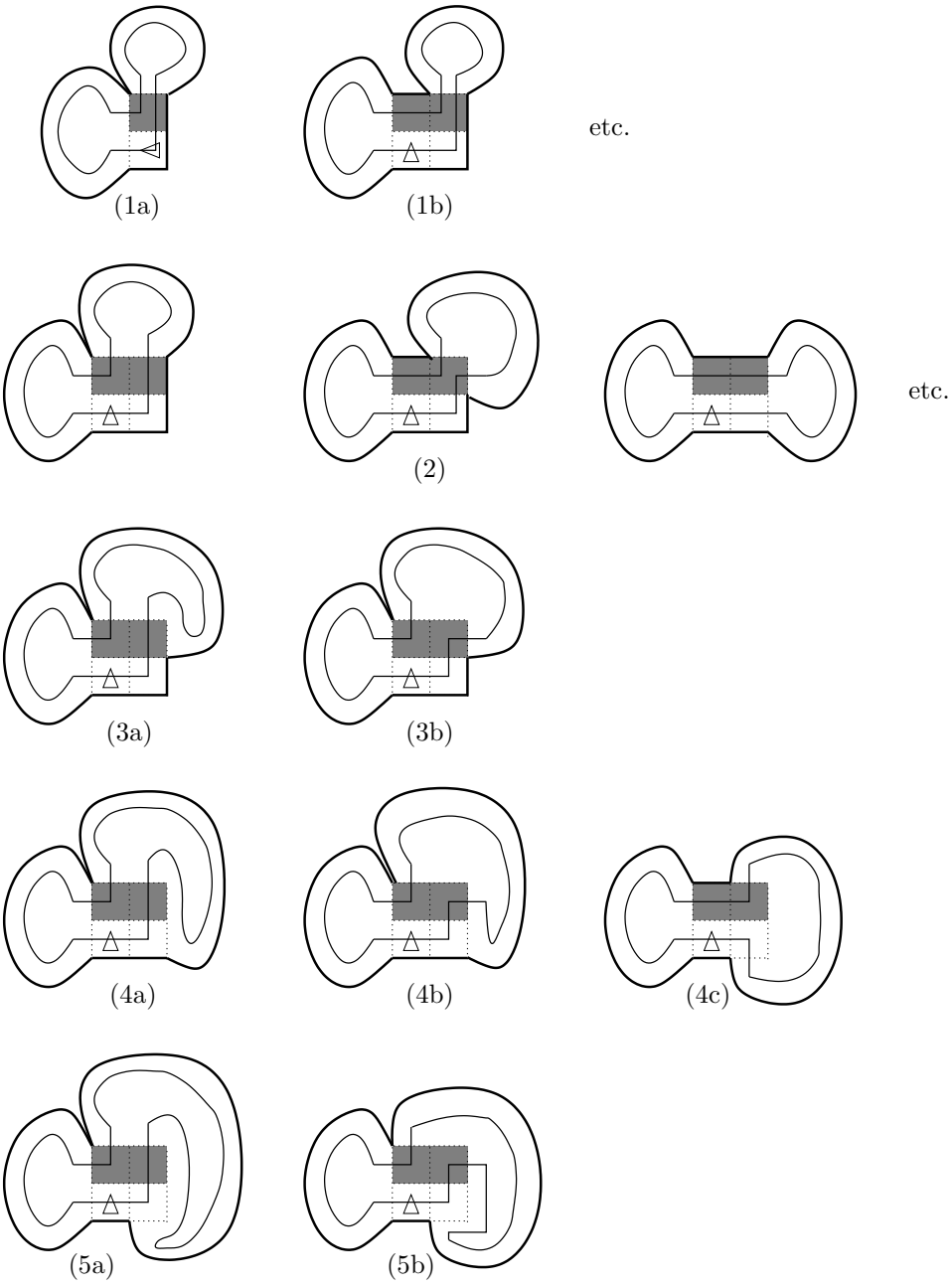
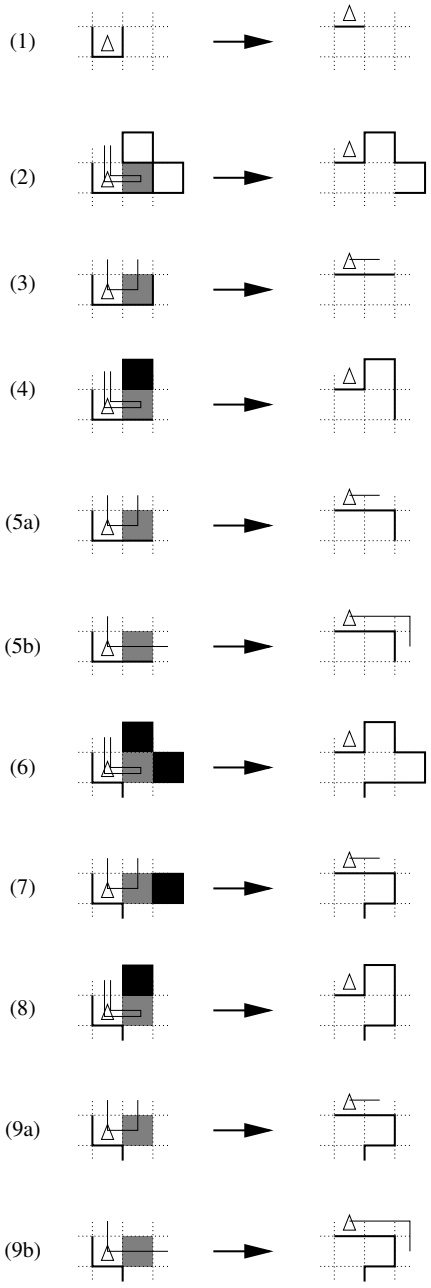
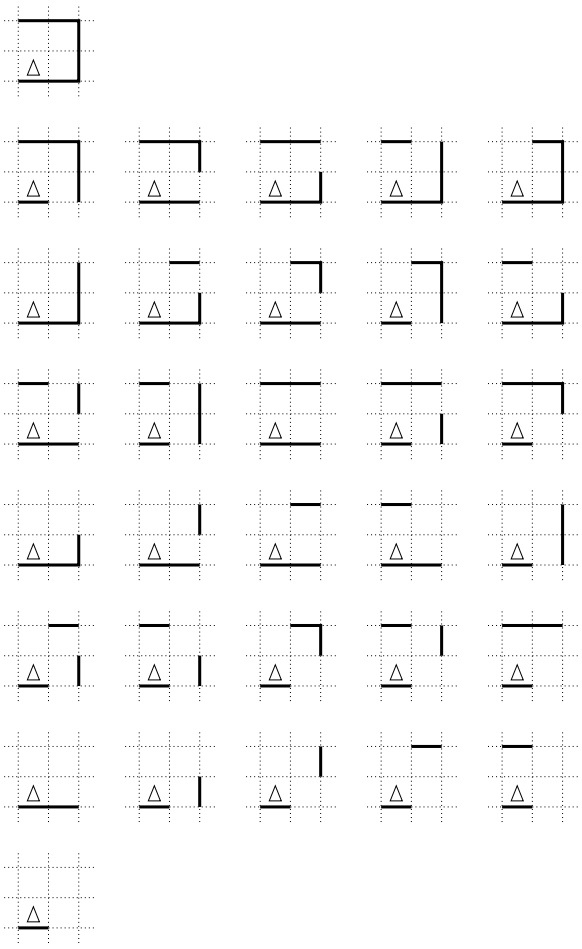



Figure 2.29: Some possible cases of polygon splits.



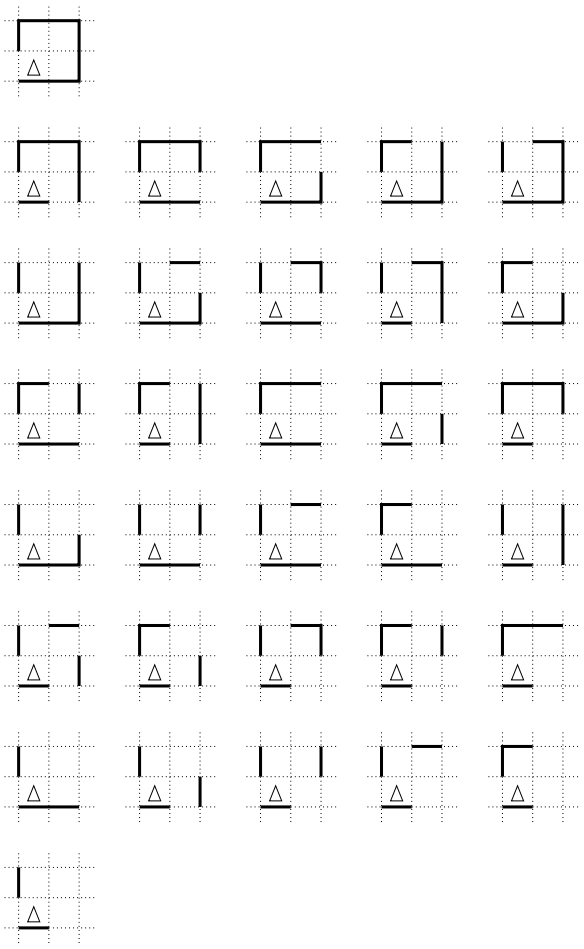
| | $\Delta S$ | $\Delta C$ | $\Delta E/2$ | $\Delta H$ | $G$ | |
|---|---|---|---|---|---|---|
| (1) | +2 | +1 | +1 | 0 | 0 | |
| (2) | +4 | +2 | +2 | 0 | 0 | |
| (3) | +2 | +2 | +1 | 0 | +1 | |
| | +2 | +2 | +1 | +1 | +2 | |
| (4) | +4 | +2 | +1 | +1 | 0 | |
| (5a) | +2 | +2 | 0 | 0 | 0 | |
| | +2 | +2 | 0 | +1 | +1 | |
| (5b) | 0 | +2 | 0 | 0 | +2 | |
| (6) | +4 | +2 | +1 | +1 | 0 | (*) |
| (7) | +2 | +2 | 0 | +1 | +1 | (*) |
| (8) | +4 | +2 | 0 | +1 | −1 | (*) |
| (9a) | +2 | +2 | −1 | 0 | −1 | (*) |
| | +2 | +2 | −1 | +1 | 0 | |
| (9b) | 0 | +2 | −1 | 0 | +1 | (*) |

Figure 2.30: Cell configurations for forward steps.



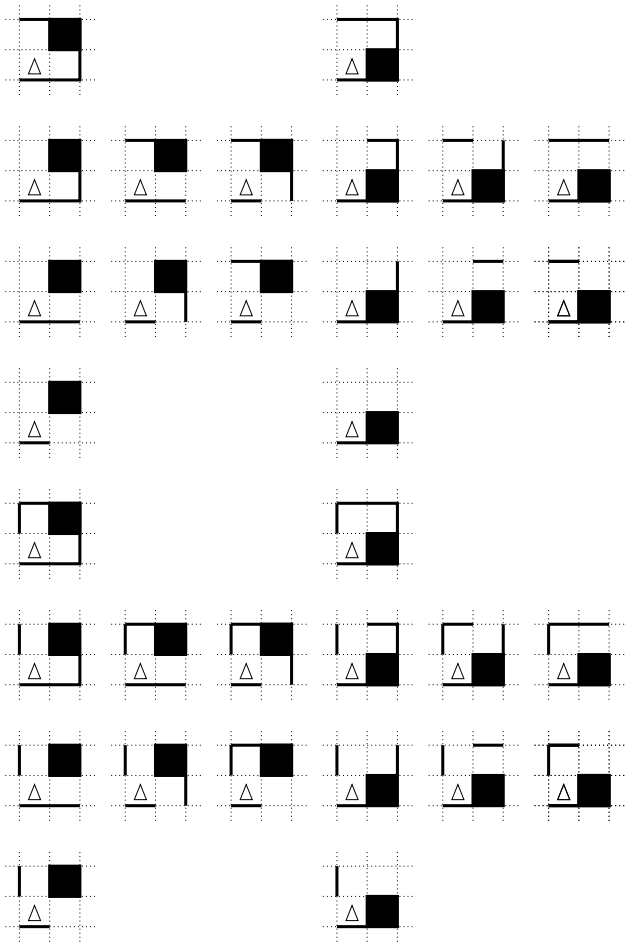
| $\Delta S$ | $\Delta C$ | $\Delta E/2$ | $\Delta H$ | $G$ |
|---|---|---|---|---|
| +4 | +4 | +2 | 0 | +2 |
| +4 | +4 | +1 | 0 | +1 |
| +4 | +4 | +1 | +1 | +2 |
| +4 | +4 | 0 | 0 | 0 |
| +4 | +4 | 0 | +1 | +1 |
| +4 | +4 | 0 | +2 | +2 |
| +4 | +4 | −1 | 0 | −1 |
| +4 | +4 | −1 | +1 | 0 |
| +4 | +4 | −1 | +2 | +1 |
| +4 | +4 | −2 | 0 | −2 |
| +4 | +4 | −2 | +1 | −1 |
| +4 | +4 | −2 | +2 | 0 |
| +4 | +4 | −2 | +3 | +1 |
| +4 | +4 | −3 | 0 | −3 |
| +4 | +4 | −3 | +1 | −2 |
| +4 | +4 | −3 | +2 | −1 |
| +4 | +4 | −3 | +3 | 0 |

Figure 2.31: Possible configurations for steps to the left (1).



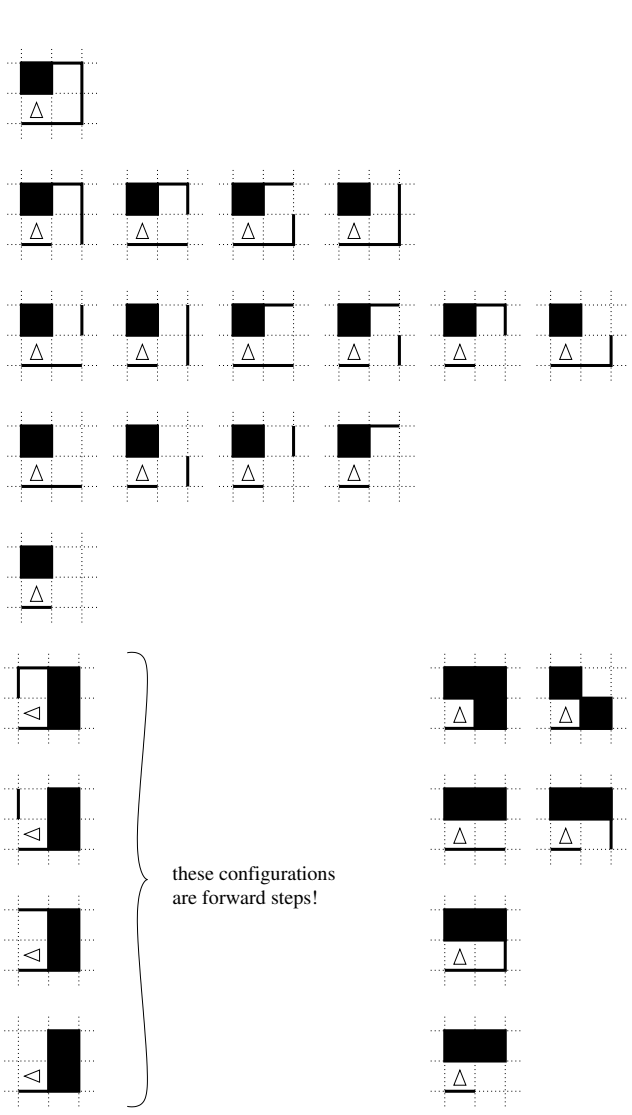
| $\Delta S$ | $\Delta C$ | $\Delta E/2$ | $\Delta H$ | $G$ |
|---|---|---|---|---|
| +6 | +4 | +3 | 0 | +1 |
| +6 | +4 | +2 | +1 | +1 |
| +6 | +4 | +1 | +1 | 0 |
| +6 | +4 | +1 | +2 | +1 |
| +6 | +4 | 0 | +1 | −1 |
| +6 | +4 | 0 | +2 | 0 |
| +6 | +4 | 0 | +3 | +1 |
| +6 | +4 | −1 | +1 | −2 |
| +6 | +4 | −1 | +2 | −1 |
| +6 | +4 | −1 | +3 | 0 |
| +6 | +4 | −2 | +1 | −3 |
| +6 | +4 | −2 | +2 | −2 |
| +6 | +4 | −2 | +3 | −1 |

Figure 2.32: Possible configurations for steps to the left (2).



| $\Delta S$ | $\Delta C$ | $\Delta E/2$ | $\Delta H$ | $G$ |
|---|---|---|---|---|
| +4 | +3 | +2 | 0 | +1 |
| +4 | +3 | +1 | 0 | 0 |
| +4 | +3 | +1 | +1 | +1 |
| +4 | +3 | 0 | 0 | −1 |
| +4 | +3 | 0 | +1 | 0 |
| +4 | +3 | 0 | +2 | +1 |
| +4 | +3 | −1 | 0 | −2 |
| +4 | +3 | −1 | +1 | −1 |
| +4 | +3 | −1 | +2 | 0 |
| +4 | +3 | −1 | +3 | +1 |
| +6 | +3 | +3 | 0 | 0 |
| +6 | +3 | +2 | +1 | 0 |
| +6 | +3 | +1 | +1 | −1 |
| +6 | +3 | +1 | +2 | 0 |
| +6 | +3 | 0 | +1 | −2 |
| +6 | +3 | 0 | +2 | −1 |
| +6 | +3 | 0 | +3 | 0 |

Figure 2.33: Possible configurations for steps to the left (3).



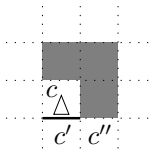

| $\Delta S$ | $\Delta C$ | $\Delta E/2$ | $\Delta H$ | $G$ |
|---|---|---|---|---|
| +6 | +3 | +3 | 0 | 0 |
| +6 | +3 | +2 | +1 | 0 |
| +6 | +3 | +1 | +1 | −1 |
| +6 | +3 | +1 | +2 | 0 |
| +6 | +3 | 0 | +1 | −2 |
| +6 | +3 | 0 | +2 | −1 |
| +6 | +3 | 0 | +3 | 0 |
| +6 | +3 | −1 | +1 | −3 |
| +6 | +3 | −1 | +2 | −2 |
| +6 | +3 | −1 | +3 | −1 |
| +2 | +1 | +1 | 0 | 0 |
| +4 | +2 | +1 | +1 | 0 |
| +4 | +2 | +2 | 0 | 0 |
| +4 | +2 | 0 | +1 | −1 |

Figure 2.34: Possible configurations for steps to the left (4).

Next, we want to discard the assumption we made in Lemma 2.14.

**Lemma 2.15** *The assumption that the cell behind and right hand to the robot's position is blocked can be violated only in the robot's initial position; that is, in $P_{0,0}$ and not in any $P_{k,i}$ with $k + i > 0$.*

**Proof.**




Consider a robot located in some cell $c$ with no wall behind and right hand to its position, and w. l. o. g. *dir* ='north'. If this is not the robot's initial position, but the position $s_{k,i}$ of a polygon $P_{k,i}$ occurring in the successive decomposition of the polygon, the robot must have entered the cell $c$ from the cell $c'$ below $c$ in the polygon $P_{k,i-1}$. If the cell $c''$ right to $c'$ is not a hole in $P_{k,i-1}$, $c''$ would be a reserved cell, and, thus, it would be removed together with $c'$ in the step from $P_{k,i-1}$ to $P_{k,i}$. Consequently, when the robot has reached $c$, there would be a hole behind and right hand to its current position. □

**Lemma 2.16** *The number of steps, $S$, used to explore a polygon with $C$ cells, $E$ edges and $H$ holes, is bounded by*

$$S \leq C + \frac{1}{2}E + H + 2L - 2,$$

*where $L$ denotes the number of the robot's left turns.*

**Proof.** Lemma 2.15 shows that the assumption in Lemma 2.14 can be violated only in the robot's initial position. On the other hand, we have seen in the proof of Lemma 2.14 that all cases that violate the assumption incur the costs of just one additional step. □

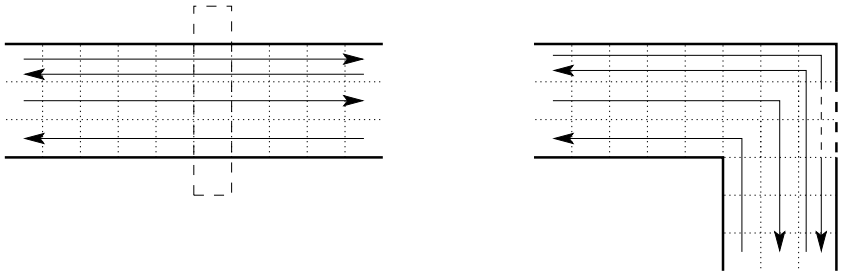



Figure 2.35: Corridors of odd width.

Our bound still depends on the number of left turns the robot makes while exploring the polygon. To give a bound that does not depend on the strategy, we introduce another property of grid polygons, a measure

to distinguish rather flat polygons from winded polygons; let us call it the *sinuosity* of $P$. Our motivation for introducing this property is the following observation: The robot may walk $n+1$ times through a corridor of width $n$, $n$ odd, see Figure 2.35(i). The costs for this extra walk are covered by the corridor walls; more precisely, for every double visit we charge *two* polygon edges and get the addend $\frac{1}{2}E$ in the upper bound. If there is a left turn in the corridor, there are not enough boundary edges for balancing the extra walk. Figure 2.35(ii) shows a corridor of width 3 with a left turn. The steps shown with dashed lines cannot be assigned to edges, so we have to count the edges shown with dashed lines to balance the number of steps. We define two types of sinuosities, the clockwise and the counterclockwise sinuosity. Because CellExplore follows the left-hand rule, our strategy depends on the clockwise sinuosity. A similar strategy that follows the right-hand rule would depend on the counterclockwise sinuosity.

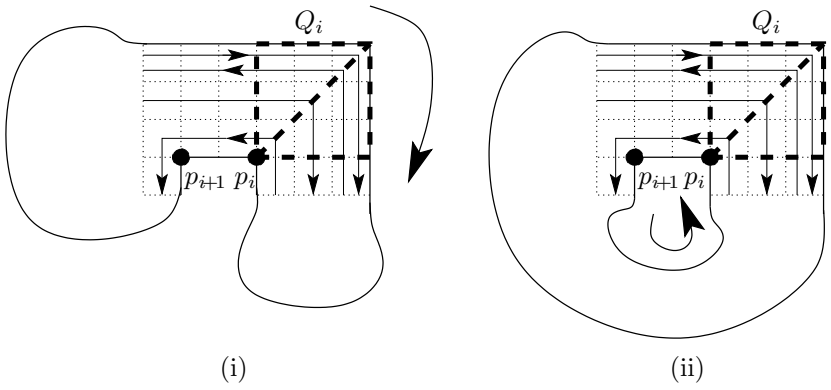



Figure 2.36: Contributions to $W_{\text{cw}}$ by (i) the outer boundary, (ii) inner boundaries.

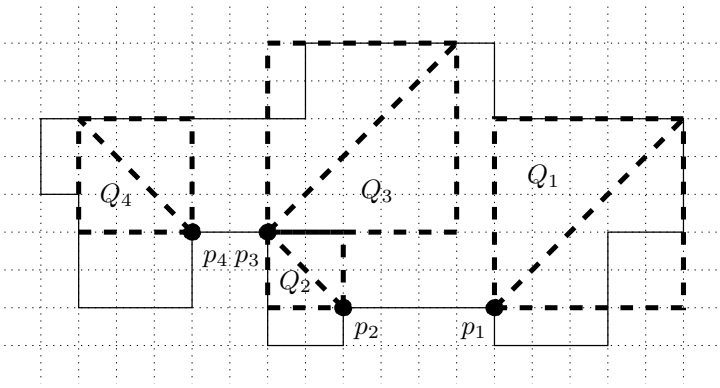



Figure 2.37: Reflex vertices $p_i$ and the corresponding squares $Q_i$. $W_{\text{cw}} = q'_1 + q'_3 = 8, W_{\text{ccw}} = q'_4 = 2$.

**Definition 2.17** Let the *clockwise sinuosity*, $W_{\text{cw}}$, and the *counterclockwise sinuosity*, $W_{\text{ccw}}$, of a grid polygon $P$ be defined as follows: We trace the boundary of $P$—the outer boundary clockwise, the boundaries of the holes inside $P$ counterclockwise—, and consider every pair, $p_i$ and $p_{i+1}$, of consecutive reflex vertices, see Figure 2.36.

We trace the angular bisector between the two edges incident to $p_i$ inside $P$ until it hits the boundary of $P$. The resulting line segment defines the diagonal of a square, $Q_i$, see Figure 2.37.[10] Let $q_i$ be the width of $Q_i$, analogously with $q_{i+1}$.

Because the robot needs some further steps only in odd corridors, we count only odd squares:

$$q'_i := \begin{cases} q_i - 1, & \text{if } q_i \text{ is odd} \\ 0, & \text{if } q_i \text{ is even} \end{cases}.$$

The need for additional edges may not only be caused by reflex vertices, but also by the start cell, see Figure 2.38(ii). Thus, we consider the squares $Q_{s_{\text{cw}}}$ and $Q_{s_{\text{ccw}}}$ from the start cell in clockwise and counterclockwise direction, respectively. Let $q'_{s_{\text{cw}}}$ and $q'_{s_{\text{ccw}}}$ be defined analogously to $q'_i$. Now, we define the clockwise sinuosity $W_{\text{cw}}$ and the counterclockwise sinuosity $W_{\text{ccw}}$ as

$$W_{\text{cw}} := q'_{s_{\text{ccw}}} + \sum_{i \geq 1} q'_{2i-1}, \quad \text{and} \quad W_{\text{ccw}} := q'_{s_{\text{cw}}} + \sum_{i \geq 1} q'_{2i}.$$

Figure 2.38 shows two examples for the definition of $W_{\text{cw}}$. Note that in (i) only one reflex vertex contributes to $W_{\text{cw}}$, and every edge we count here is needed.

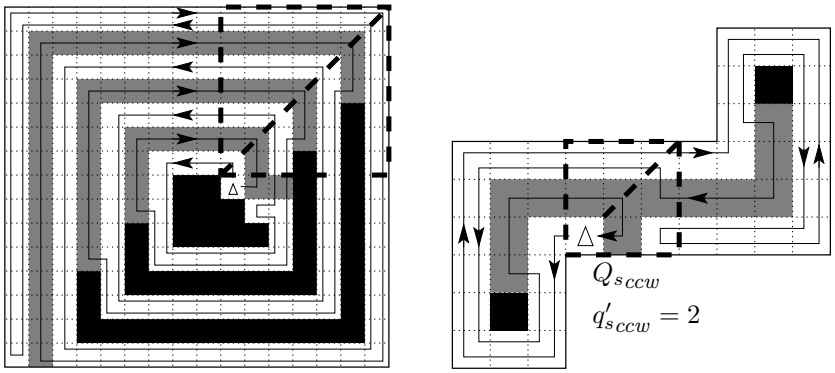



Figure 2.38: Examples for the definition of $W_{\text{cw}}$: (i) A polygon with $C = 193, \frac{E}{2} = 78, H = 3, W_{\text{cw}} = 6, S = 284$ (the bound for $S$ is exactly achieved), (ii) the start cell contributes to $W_{\text{cw}}$, too ($C = 46, \frac{E}{2} = 23, H = 2, W_{\text{cw}} = 2, S = 74$).

[10] We can construct $Q_i$ by "blowing up" a square around the cell in $P$ that touches the boundary of $P$ in $p_i$ until the corner of $Q_i$ opposite to $p_i$ hits the polygon's boundary.

With the definition of $W_{\text{cw}}$, we can give our final result:

**Theorem 2.18** *Let $P$ be a grid polygon with $C$ cells, $E$ edges, $H$ holes, and clockwise sinuosity $W_{cw}$. CellExplore explores $P$ using no more than*

$$S \leq C + \frac{1}{2}E + W_{cw} + 3H - 2$$

*steps. This bound is tight.*

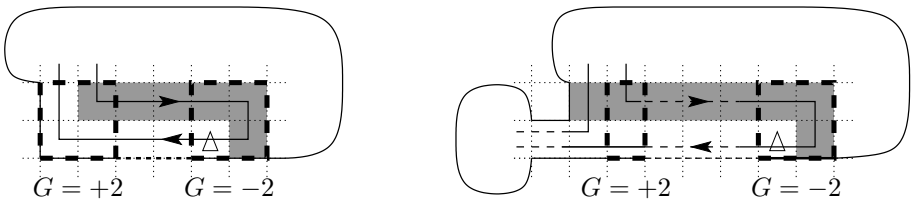



Figure 2.39: Left turn followed by (i) a right turn and (ii) a reduction.

**Proof.** We need some global arguments to charge the costs for a left turn to properties of $P$. So let us examine, which configurations may follow a left turn (after some forward steps):

- A right turn follows the left turn, see Figure 2.39(i). We gain +2 steps per right turn, so the possible costs of −2 for this left turn are covered.
- An obstacle follows the left turn. We can charge the obstacle with the costs for the left turn and get a factor of 3 for the number of obstacles. Every obstacle is charged at most once, because when the successive decomposition reaches the obstacle for the first time, the obstacle disappears; that is, the hole merges with the outer boundary.
- A reduction follows the left turn, see Figure 2.39(ii). Later in this section, we show that a reduction covers the costs of a left turn.
- Another left turn follows the observed left turn. In this case, there is no other property of $P$ to be charged with this costs but the sinuosity $W_{\text{cw}}$, this follows directly from the definition of $W_{\text{cw}}$.

In the case of a reduction following a left turn we observe the number, $d$, of forward steps between the left turn and the reduction, as well as the cell marked with $b$, and—if $d \geq 1$—the cell marked with $a$ in Figure 2.40(i). If $b$ is blocked, $b$ is either part of an obstacle inside the polygon or it is outside the polygon. In the first case, we charge the obstacle with the costs of the left turn as described earlier. In the second case we have a polygon split that

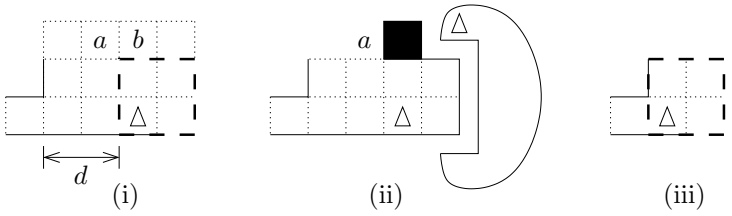



Figure 2.40: (i) A reduction follows a left turn, (ii) the left turn causes a polygon split, (iii) no forward steps between the left turn and the reduction ($d = 0$).

leaves us with a left turn that incurs no costs, see Figure 2.40(ii) and the first line of Figure 2.31. The same holds for the cell marked with $a$ if there is at least one forward step between the left turn and the reduction (i. e., $d \geq 1$). Therefore, we assume that $a$ and $b$ are free cells in the following.

If the reduction follows immediately after the left turn ($d = 0$), see Figure 2.40(iii), we have one of the left turns shown in Figure 2.32, Figure 2.34 or the lower half of Figure 2.33. In any of these cases we have either a positive balance or we meet an obstacle ($\Delta H > 0$) and charge the costs for the left turn to the obstacle as earlier.

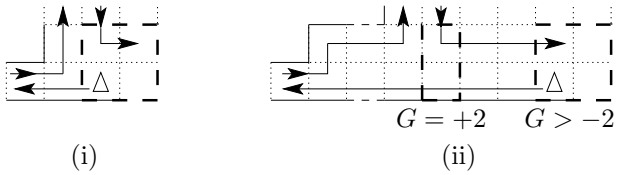



Figure 2.41: If $a$ and $b$ are free cells we gain 2. (i) $d = 1$, (ii) $d \geq 1$.

If there is one forward step between the left turn and the reduction ($d = 1$), the robot enters the $2 \times 2$ block of cells of the left turn not from the same side as it left it, because $a$ and $b$ are polygon cells. In Figure 2.41(i) the robot leaves the block to the left but enters it from above. This situation is described in Figure 2.28 on page 46 and reduces the costs for the left turn by 2, so the balance is either zero or positive. If there is more than one forward step ($d > 1$), we have either the same situation as in the preceding case, or the reduction from a corridor of width $\geq 3$ to a corridor of width $\leq 2$ shifts to the left, see Figure 2.41(ii), and eventually we reach a forward step as shown in Figure 2.30(5b) that gains +2 and covers the costs for the left turn.

Altogether, we are able to charge the costs for every left turn to other properties, which proves our bound.

Figure 2.38 and Figure 2.42 show nontrivial examples (i. e., $H \neq 0$ and $W_{\text{cw}} \neq 0$) for polygons in which the bound is exactly achieved. □

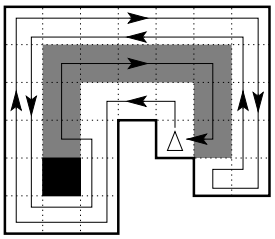


Figure 2.42: Polygon with $C = 34, \frac{E}{2} = 17, H = 1, W_{\text{cw}} = 2, S = 54 = C + \frac{1}{2}E + 3H + W_{\text{cw}} - 2$.

## 2.4 Concluding Remarks

In this section, we consider some further aspects concerning cellular environments. We briefly discuss other strategies for the exploration of grid polygons with holes and strategies in three-dimensional environments, and present a simulation environment for exploration strategies. Finally, we introduce another model for robots in grid polygons.

### 2.4.1 CellExplore with Optimized Return Path

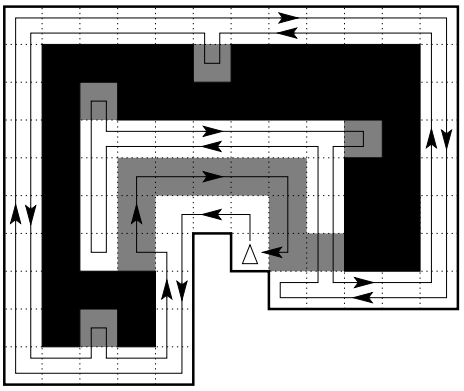


Figure 2.43: A polygon with $C = 69, \frac{E}{2} = 52, H = 1, W_{\text{cw}} = 2, S = 124 = C + \frac{1}{2}E + 3H + W_{\text{cw}} - 2$. The return path in this polygon cannot be shortened.

A straightforward improvement to the strategy CellExplore is to use in the backward mode the shortest path—on the cells known so far—to the first cell on the stack that is unexplored or has unexplored neighbors instead of walking back using every reserved cell, see the first improvement of DFS. From a practical point of view, this improvement is very reasonable, because the performance of the strategy increases in most environments. Unfortunately, the return path (i. e., the path walked in the backward mode) is no longer determined by a local configuration of cells. Instead, we need a global view, which complicates the analysis of this strategy. However, there

are polygons that force this strategy to walk exactly the same return path as CellExplore without any optimization, see Figure 2.43, so this idea does not improve the worst case performance, and the upper bound for the number of steps is the same as in Theorem 2.18.

### 2.4.2 The Solution of Gabriely and Rimon

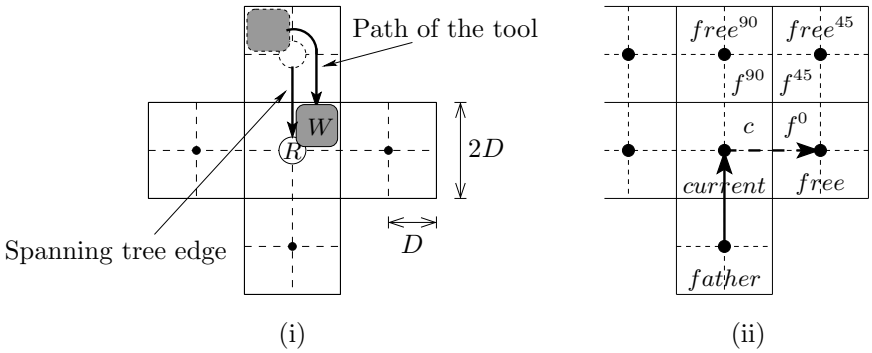



Figure 2.44: (i) The model of Gabriely and Rimon: the robot $R$, the tool $W$, cells, and $2D$-cells, (ii) avoiding turns in Scan-STC.

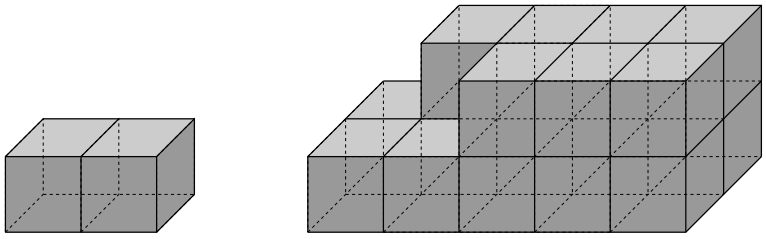
Parallel to our work, Gabriely and Rimon [63] introduced another solution for the exploration of grid polygons with holes. They use a slightly different model: A robot is equipped with a tool of size $D \times D$ that rotates around the robot. Four cells in a $2 \times 2$ block are combined to a so-called $2D$-cell, see Figure 2.44(i). With this, the robot moves from midpoint to midpoint of a $2D$-cell, constructing a spanning tree on the graph of all $2D$-cells, while the tool moves keeping the spanning tree edges on its right side. If cells on one side of the spanning tree edge are blocked, the tool changes to the other side of the spanning tree until the original side becomes unblocked. The strategy *Spiral-STC* (Spanning Tree Covering) uses the right-hand rule for the construction of spanning tree edges; the strategy *Scan-STC* tries avoid horizontal edges and thus to reduce the number of turns. Whenever the Spiral-STC strategy is about to add a horizontal spanning tree edge from the current $2D$-cell to the $2D$-cell *free*, the Scan-STC strategy checks the cells labeled $f^{45}$ and $f^{90}$ in Figure 2.44(ii). If both cells are free, the cell $f^0$ can be reached with a vertical step from $f^{45}$, and the robot continues its path with a step from *current* to $free^{90}$ avoiding the horizontal step from *current* to *free*. Remark that this technique can also be applied to CellExplore. However, the scanning strategies require an extension of the robot's abilities, because it is necessary to check the cell $f^{45}$, which is not adjacent to the current cell.

Gabriely and Rimon showed an upper bound of $C + B$ steps for their strategies, where $C$ denotes the number of cells and $B$ the number of bound-

ary cells (i. e., the number of free cells that touch an obstacle cell). In polygons without any $2 \times 2$ block of free cells, $C + B$ is equal to our bound from Theorem 2.18, but in polygons with wider areas our bound is considerably smaller than $C + B$. In the worst case, Gabriely and Rimon charge roughly all edges,[11] whereas our bound uses only slightly more than half the number of edges.

### 2.4.3 Exploring Three-Dimensional Environments

Modifying our strategies to explore three-dimensional cellular environments is an interesting problem. A cell in three dimensions is a cube; each of them having six neighboring cells and 26 touching cells. With this, a robot located in the cell $(x, y, z)$ can move to six directions. We call a move to the positive $X$ $(Y, Z)$ direction a move to the east (north, up; respectively) and in negative direction a move west (south, down; respectively).



(i) (ii)

Figure 2.45: Cubical environments, (i) $C = 2, F = 10, E = 16$, (ii) $C = 17, F = 46, E = 39$.

Analogously to the number of cells and edges in a grid polygon, we have in the three-dimensional case the number, $C$, of cubic cells and the number, $F$, of faces between a free and a blocked cell. Additionally, we have the number of outer edges, $E$, that is the total number of all edges, where a free cell meets *three* blocked cells.

Obviously, the lower bounds from Theorem 2.2 and Theorem 2.3, see Section 2.1, still hold in the three-dimensional case. We can use the same construction, replacing every square by a cube.

**Corollary 2.19** *The competitive complexity of exploring an unknown cubical environment with obstacles is equal to 2, and every strategy needs at least* $\frac{7}{6}C$ *steps in a simple polyhedron*[12] *with* $C$ *cells.*

[11] By Lemma 2.5 only four edges in the outer boundary are not charged. However, there a cases in which even $B > E$ holds, because a hole with $E'$ edges has—by the same lemma—$E' + 4$ boundary cells.

[12] Following [188], we call a grid polyhedron *simple*, if it is topologically equivalent to a sphere.

**Algorithm 2.4** SmartDFS-3D

```
ExploreCell(dir):
    base := current position;
    if not isSplitCell(base) then
      if (dir == 'up') or (dir == 'down') then
        // Reset direction to explore next slice
        dir := 'north'
        ExploreStep(base, 'south');
      end if
      // Left-Hand Rule:
      ExploreStep(base, ccw(dir));
      ExploreStep(base, dir);
      ExploreStep(base, cw(dir));
      ExploreStep(base, 'up');
      ExploreStep(base, 'down');
    else
      Same as above, but in different order.
    end if
```

A three-dimensional version of DFS, see Algorithm 2.1 on page 21, is rather straightforward; we simply have to consider the two additional neighbors. Note that it is not obvious how to state the left-hand rule in three dimensions. A solution is, to view a grid polyhedron as a stack of slices parallel to the $XY$ plane and to proceed in every slice using the left-hand rule. If no step following the left-hand rule is possible, we move up or down; thus, we continue with the next upper or lower slice. The two improvements to DFS—optimizing the return path as well as the detection and handling of split cells—can be applied also in three dimensions. This leads to a strategy SmartDFS-3D similar to Algorithm 2.2 on page 23 for the exploration of *simple* cellular polyhedra. Algorithm 2.4 shows the corresponding procedure *ExploreStep*; Figure 2.46[13] shows an example. After a step upward or downward we set the direction to *north*, because we want to explore the next slice. Besides, *ccw* and *cw* are not defined for the directions up and down. Note that we explicitly explore the cell to the south in this case, otherwise this cell may not be visited, see Figure 2.47(i). Remark also that we can no longer determine the layer number of a cell in its first visit, so the implementation of the split cell detection and handling is not as straightforward as in two-dimensional case.

[13] For convenience, we show only the projection of the cells into their $XY$ plane—similar to a floor plan—and omit the height of the cells. Therefore, the top and the sides of the shown cells are missing.

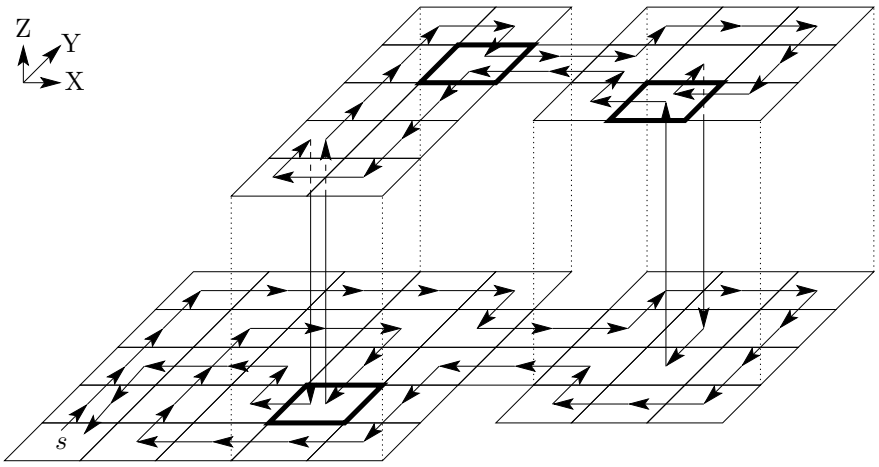



Figure 2.46: Exploring a cubical environment with SmartDFS. Split cells are highlighted.

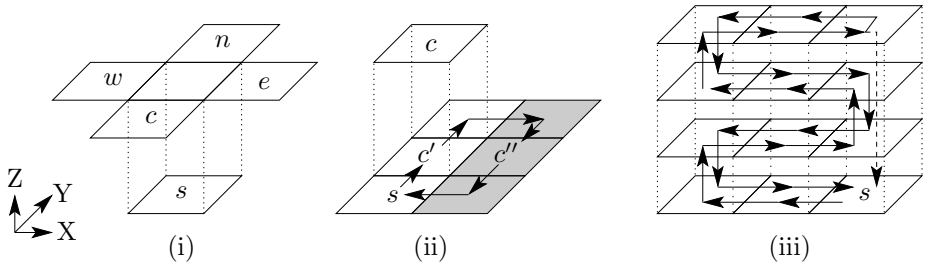



Figure 2.47: (i) Without an explicit step to the south, a robot using SmartDFS-3D starting in $s$ does not explore the cell $c$, (ii) a CellExplore-like strategy in 3D has to ensure that $c$ is visited (the shaded cells are reserved cells), (iii) a CellExplore-like strategy in 3D may perform as badly as DFS.

We conject that the upper bound for the length of a shortest path inside a grid polyhedron, compare to Lemma 2.8 on page 31, is roughly $\frac{1}{4}E$, because we can split a shortest path into parts parallel to the $X$, $Y$ and $Z$ axis. Every step is—so to speak—embedded by four edges along the corresponding axis. Thus, we conservatively conject that the upper bound for the number of steps of SmartDFS-3D is roughly $C + \frac{1}{4}E$.

The slice-wise left-hand rule can also be used to generalize CellExplore, see Algorithm 2.3 on page 39. Now, we have ensure that all cells above or below cells that are visited in the forward mode are explored. See, for example, Figure 2.47(ii): The robot uses the standard CellExplore to explore the lower plane, but misses the cell $c$. An idea to solve this problem is to use two additional markers for every cell. The marker *up* is set for cells that are visited in the forward mode, but have an unexplored neighbor above—analogously for a *down* marker. For example, in Figure 2.47(ii) the *up* marker

for $c'$ is set during the first visit of $c'$. For every visited cell, we reset the *up* marker for the cell below. In the backward mode, we check if the *up* marker for a cell adjacent to the current cell still is set. In this case, we walk to the yet unvisited cell and switch to the forward mode to start an exploration from this cell. After this, we continue with our path in the backward mode. In our example, the *up* marker in $c'$ is still set when we reach $c''$, therefore we walk via $c'$ to $c$. $c$ has no unexplored neighbor, so we return immediately to $c''$. Dienelt [45] successfully implemented this idea and gave a detailed description of the strategy.

Another solution is to push not only the reserved cells onto the stack, but also every cell that is explored in the forward mode, and to optimize the path in the backward mode as described in Section 2.4.1. This solves also the problem that the first idea is rather inefficient, if it is not possible to reserve cells in the same slice, but in the slice above or below. Consider an environment that is only one slice parallel to the $YZ$ plane, see Figure 2.47(iii): The first strategy behaves exactly like DFS, whereas the second strategy is able to shorten the return path (dashed line). However, from our experience with two dimensions we conject that this strategy is rather difficult to analyze. Even for the first strategy it not easy to estimate the number of steps. In the analysis of CellExplore we have seen several cases, in which we had to charge a hole inside the polygon, but we had to do this at most once for every hole. In the three-dimensional case, a single hole can appear in many slices parallel to the $XY$ plane, and we may have to charge the hole for every such slice. Thus, it not clear whether we can adapt the proof technique for CellExplore at all.

### 2.4.4 A Simulation Environment

Handel et al. [77] developed a simulation software for exploration strategies in grid polygons, including both SmartDFS and CellExplore, see Figure 2.48. The Java applet consists mainly of two parts. First, the polygon editor allows to create and to modify arbitrary polygons, or to generate polygons randomly, so the user is not restricted to a few pre-coded examples, but can try out any desired polygon. After switching from the editor to the polygon explorer, the user can explore the polygon manually, observe the behavior and performance of several strategies, and even compete against the lower bounds that we have shown in Theorem 2.2 and Theorem 2.3, see the *adversary* selector in Figure 2.48

It was a great help to have a tool that takes on the annoying and error-prone task of counting cells and edges, and, more important, allows to experiment with several versions of CellExplore, see the *options* panel in Figure 2.48. For example, we studied a strategy where the cells right to the path in the forward mode are reserved and used for the return path, even if they are already reserved by another cell in the forward mode, see

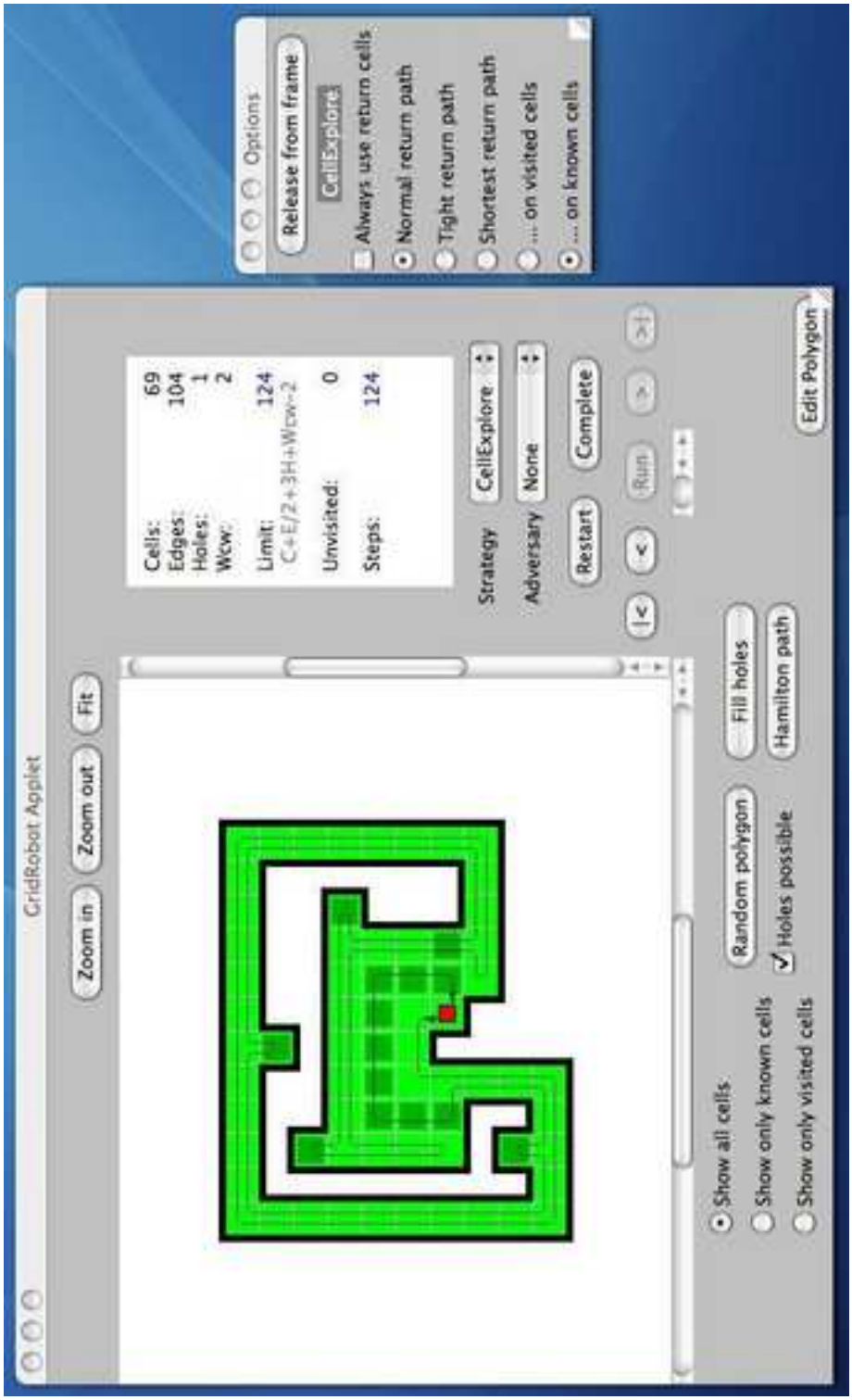
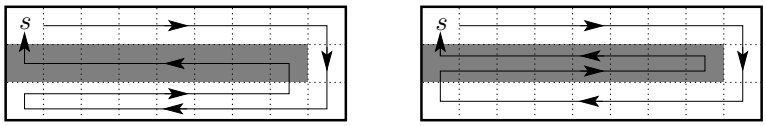



Figure 2.48: The GridRobot applet and the option panel for CellExplore.




(i) (ii)

Figure 2.49: (i) CellExplore vs. (ii) the version *use reserved cells always*.

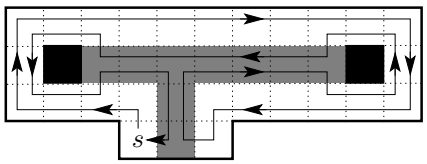



Figure 2.50: The version *use reserved cells always* exceeds the bound from Theorem 2.18 by one step. $C = 34, \frac{E}{2} = 19, H = 2, W_{\text{cw}} = 0, S = 58$.

Figure 2.49. With some experiments, we found a polygon that exceeded our upper bound, and therefore we rejected this version, see Figure 2.50. Moreover, the tool helped to state and falsify some conjectures about the strategy's performance. For example, we tried to show a conjectured upper bound of $C + \frac{1}{2}E + 3H - 3$ for some time, but eventually found a counterexample to this conjecture. Some more tries led to the understanding that it is necessary to take the additional edges in Definition 2.17 (sinuosity) into account. Of course, even the best simulator is no replacement for reasonable proofs, it just gives some valuable insights.

Recently, Dienelt [45] developed a simulation environment for exploration strategies in three-dimensional grids that may help developing further strategies in three dimensions and finding an upper bound for them.

A simulation environment does not only serve for research purposes. It also helps others to understand the strategies. We taught the exploration of (grid-)polygons in lectures about robot motion planning [101, 102]; the Java applet allowed our students to watch the strategies in different polygons and to try out the lower bounds. The feedback was throughly positive: The students, who used the applet, reported that it helped to understand the subject. The same holds for other Java applets that show other algorithms or structures in robot motion planning or—more generally—in computational geometry, see the *Geometry-Lab* website [66] or Icking et al. [91].

Having a simulation environment is one benefit, writing it is another. To write the program, the designer of an algorithm has to concretize every line of pseudocode and every vaguely formulated idea. This gives a much deeper insight into the problem. Examples for this are the detection of split cells in SmartDFS, or the development of the lower bound for simple grid polygons.

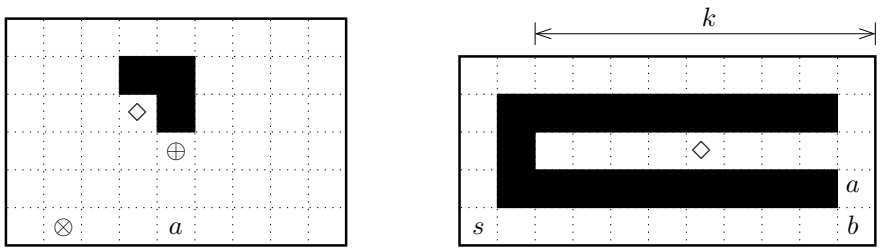
Java seems to be a good choice for the implementation of a simulation software. The language fulfills all requirements to a modern programming language like object orientation and strong typing. Java comes with a comprehensive library of methods, data structures, and interfaces to a graphical oriented window system. Moreover, Java is platform independent, so Java applets can be downloaded and run in the user's internet browser, requiring only the installation of a Java plugin that is bundled with almost every currently available internet browser under every reasonable computer and operating system. This allows to make Java applets available to almost every user without having to compile or even to install the program.

Recently, with Macromedia Flash [139] appeared an interesting alternative to Java. Flash offers the same platform independency as Java, Flash programs—called *movies* in the Flash community—require only the Flash player from Macromedia, which is available for free and for a wide range of computer platforms. Although the JavaScript-based programming language is a little bit sloppier than Java, it is remarkably powerful. Especially, it is easy to create keyframe animations with Flash. A drawback is that the authoring tool for Flash movies is not free of charge in contrast to the Java Development Kit from Sun Microsystems [176].

### 2.4.5 Robots with Restricted Orientation

Another interesting model for robots moving around in cellular environments was inspired by the board game *Ricochet Robots* by Alex Randolph [156]. The game consists of four robots in different colors, several marker chips, and a game board showing some obstacles and some cells marked with different symbols. Initially, the robots are placed randomly on the board and the markers are hidden. In every turn one of the markers is drawn. The players try to figure out the smallest number of moves that are necessary to move the robot in the revealed color to the cell with the revealed symbol. The interesting part of the game are the rules to move a robot: A robot can move in one of the four directions (north, east, south, or west), but once it has chosen a direction it continues to move in this direction until it hits an obstacle or another robot. Thus, it is often necessary to move robots that serve as guides to stop the movement of another robot on an appropriate cell. See, for example, Figure 2.51(i): The task is to move the robot $\otimes$ to the cell marked with $\diamond$. To permit this movement, the robot $\oplus$ has to move to $a$, so three moves are necessary to solve the task. Of course, the player that finds the minimal number of movements wins the turn.

This model can be used for a set of robots. Each of them has a very restricted orientation. Even if the robots have a map of their environment, a robot that touches a wall knows only roughly, which wall in the environment it touches, but as soon as the robot leaves the wall it has no chance to locate itself. Therefore, it continues its movement until it hits another wall




(i) (ii)

Figure 2.51: (i) Example: the robot $\oplus$ has to move to $a$ to allow the robot $\otimes$ a movement to $\diamond$, (ii) $\lceil \frac{k}{2} \rceil + 1$ robots are necessary to reach $\diamond$.

or another robot. However, the robots are able to communicate with each other, or all of them are controlled by the same computer. Let us assume, all robots start on the same start cell, $s$.[14] Apart from the best strategy to solve Randolph's game, an interesting question is, whether there is an upper bound for the number of robots, such that every cell can be reached by at least one robot. If there are passages of width 1 in the environment, we can "trap" robots. See, for example, Figure 2.51(ii): The polygon includes a corridor of width 1, and we need $\lceil \frac{k}{2} \rceil + 1$ robots starting in $s$ before $\diamond$ is reached—one robot in $a$ and $\lceil \frac{k}{2} \rceil$ robots to "fill" the corridor from the left or from the right. Note that we need one robot located in the cell $b$ before another robot is able to reach $a$, but the robot in $b$ can be used again after $a$ is occupied.

Even if we have a simple polygon, corridors of width 1 may cause the need for an arbitrary number of robots, see Figure 2.52(i): We need also $\lceil \frac{k}{2} \rceil + 1$ robots starting in $s$ to occupy $t$. But what happens, if the polygon does not have such narrow passages? Are there still polygons that need an arbitrary number of robots, or is there an upper bound? Engels [52] showed the following theorem for general grid polygons:

**Theorem 2.20** *(Engels, 2005)*
*Given a grid polygon and a set of k Randolph robots, the problem of deciding whether one the robots is able to reach a specified target cell is NP-complete.*

For simple polygons (i. e., polygons without holes), only the following simple statement is known:

**Lemma 2.21** *Given a rectangle of size $m \times n$, $m, n > 1$, without holes, every cell can be reached using at most three Randolph robots with $O(m+n)$ moves.*

[14] We imagine the robots enter the polygon successively through a door in the wall or in the floor.

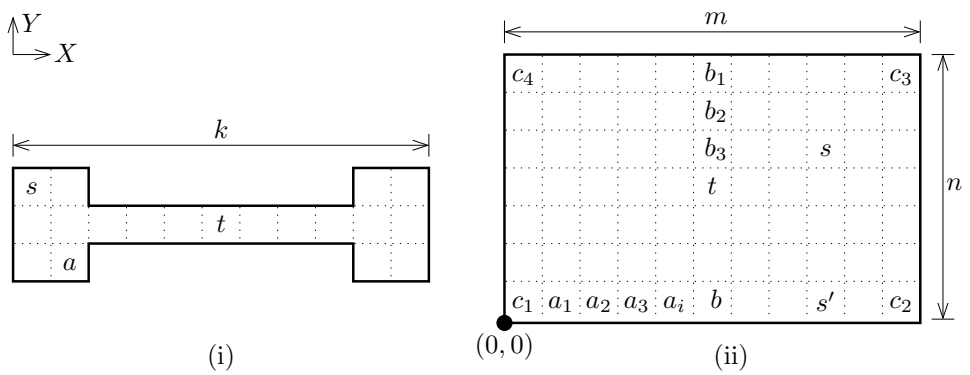



Figure 2.52: (i) $\lceil \frac{k}{2} \rceil + 1$ robots are needed to reach $t$, (ii) $t$ can be reached with $O(n+m)$ steps using 3 robots.

**Proof.** Consider an arbitrary start cell, $s$, and an arbitrary target, $t = (x, y)$, see Figure 2.52(ii). W. l. o. g. we assume that $x \leq \lceil \frac{m}{2} \rceil$ and $y \geq \lceil \frac{n}{2} \rceil$ holds; otherwise, we choose another cell than $c_1$ in the following.

We reach $t$ using the following strategy. In the first stage, we place one robot to mark the column containing $t$: The first robot moves from $s$ via $s'$ to $c_1$ with 2 moves, the second robot moves from $s$ to $a_1$. Now, the first robot moves via $c_4, c_3$ and $c_2$ to $a_2$, the second robot moves a similar path to $a_3$. This continues, until eventually one of the two robots has reached the cell $a_i$. With the help of the robot in $a_i$ we can proceed vertically to $t$: The other robot moves via $c_3, c_2$ and $b$ to $b_1$. Now, we need a third robot that moves via $s'$ and $b$ to $b_2$, and the robot located on $b_1$ is now free to move to $b_3$. We proceed in this way until we reach $t$. It is easy to see that we use no more than $4\lceil \frac{m}{2} \rceil$ moves to occupy the cells $a_i$ and $b$, and no more than $4\lceil \frac{n}{2} \rceil$ further moves to reach $t$, yielding $O(n+m)$ moves. □

### 2.4.6 Summary

In this chapter, we considered the exploration of grid polygons. For simple polygons we have shown a lower bound of $\frac{7}{6}$ and presented a strategy, SmartDFS, that explores simple polygons with $C$ cells and $E$ edges using no more than $C + \frac{1}{2}E - 3$ steps from cell to cell. Using this upper bound, we were able to show that SmartDFS is in fact $\frac{4}{3}$-competitive, leaving a gap of only $\frac{1}{6}$ between the upper and the lower bound.

On the other hand, the competitive complexity for the exploration of grid polygons with holes is 2. A simple DFS exploration already achieves this competitive factor, but, nevertheless, DFS is not the best possible strategy, because is it not necessary to visit *each* cell twice. Therefore, we developed the strategy CellExplore that takes advantage of wider areas in the polygon, and thus corrects the weakness in the DFS strategy.

Finally, we briefly discussed the exploration of three-dimensional environments, but we left the performance of SmartDFS-3D and CellExplore-3D as open questions. More further work might be done with the robot model introduced in the previous section, and in the exploration of environments that are composed of cells in a form other than a square (i. e., a triangle or a hexagon).

# Chapter 3

# Searching with Error-Prone Robots

In the field of robot motion planning many aspects are studied and solved by researchers from different communities. From a theoretical point of view, many tasks are well understood; in the introduction we already mentioned, for instance, the polygon exploration strategies [42, 84], and several kinds of search strategies [64, 11, 113, 93, 164].

Theoretical correctness results and performance guarantees often suffer from idealistic assumptions; for example, the assumption that the robot is point shaped or error free. Because of these assumptions, given performance bounds may not be achieved in "real" environments, or—in the worst case—a correct implementation of the given algorithms is impossible.

On the other hand, practitioners analyze correctness and performance mainly statistically or empirically, see, for example, [12, 13, 119, 120, 127, 186, 189]. Often, the performance is given only in terms of the running time in milliseconds for several examples, and expressive statements concerning asymptotic running time or competitive factors are missing.

Thus, there seems to be a kind of gap between the practically and the theoretically oriented schools—unfortunately. To close this gap it may be useful to investigate how theoretically well-analyzed algorithms with idealistic assumptions behave if those assumptions cannot be fulfilled. More precisely, can we design appropriate models for errors in sensors and motion, and can we incorporate these error models into the theoretical analysis?

In this chapter we consider two examples of theoretically well-understood strategies: the Pledge algorithm for escaping a maze, and the doubling strategy for finding a point on a line. We examine sources of errors for this strategies. Further, we show how errors may be modeled and analyze the strategies with respect to possible errors.

For other approaches to robust strategies and error handling see, for example, [130, 144, 145, 146, 47, 138, 7, 40, 174, 175, 126, 28, 190, 35, 21, 22].

## 3.1 Leaving an Unknown Maze

As a first approach, we consider a rather simple and theoretically well-understood strategy: the Pledge algorithm, see Abelson and diSessa [1], and Hemmerling [79]. Given an unknown polygonal scene—the maze—the robot has to leave the maze using nothing else than a touch sensor, the ability to measure its turning angles, and a very limited amount of memory.

The Pledge algorithm assumes that the robot is able to move a straight line between the obstacles and to count its turning angles correctly. Gritzmann [70] remarked that it would be interesting to know how the Pledge algorithm behaves, if those assumptions cannot be fulfilled. Of course, if the robot can make arbitrary big mistakes, there are always environments in which the robot is hopelessly trapped. But what if the robot's errors are small enough? Are there upper bounds for measuring errors? We investigate which conditions must hold to ensure that a robot can leave an unknown maze with a Pledge-like algorithm.

An application of this problem is a robot that has to leave an unknown maze using mainly a compass device. The compass readings may be faulty due to disturbing magnetic fields or electrical deviations. We show that we can solve this problem using the Pledge algorithm and give an upper bound for the error in the compass readings, see Section 3.1.3.1.

We assume that the robot is not aware of making any errors; it always believes that its movement and angle counting is correct.

### 3.1.1 The Pledge Algorithm

**Algorithm 3.1** Pledge

```
ω := 0.
repeat
  repeat
    Move in direction ω in the free space.
  until Robot hits an obstacle.
  repeat
    Follow the wall in counter-clockwise direction.
    Count the overall turning angle in ω.
  until Angle Counter ω = 0.
until Robot is outside the maze.
```

We have given a maze with polygonal obstacles in the plane, and a robot that is able to recognize and follow a wall in a specified direction (w. l. o. g. counter-clockwise) and to count the turning angles. The realization of these abilities depends on the hardware of a specific robot. For example, the turning angles may be counted by odometry, by measuring angles along the

walls with sensors, or using a compass device. Other abilities for orientation and navigation are not required; especially, it is not necessary that the robot can build a map of its environment or set landmarks.

The task of leaving an unknown maze can be solved using the well-known Pledge algorithm, see Algorithm 3.1,[1] which performs only two types of movements: Either the robot follows the wall of an obstacle counting its turning angles, or the robot moves in a fixed direction through the free space between the obstacles. The latter task always starts at vertices of the obstacles when the angle counter reaches a predefined value. We assume that the robot receives—somehow or other—a signal of success as soon as it leaves the maze.

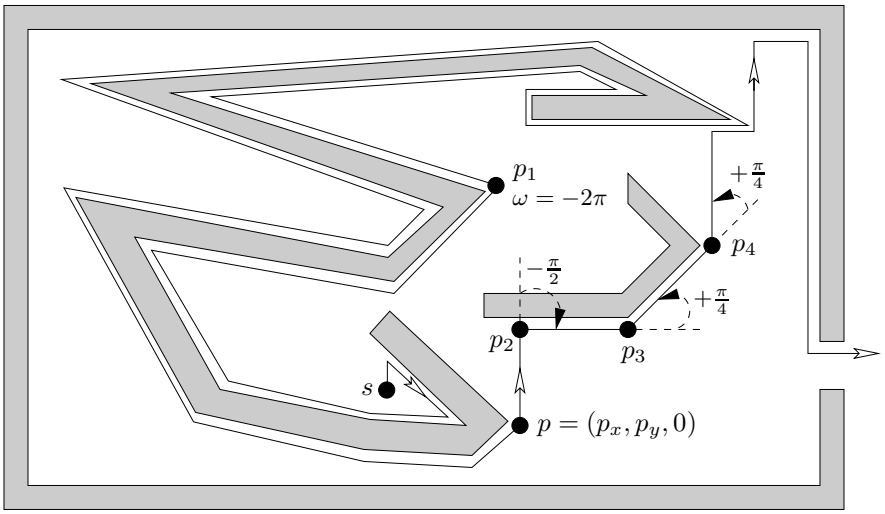
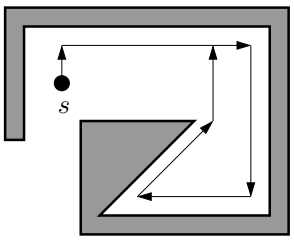



Figure 3.1: The path of the Pledge algorithm.

In the idealized setting the robot is error-free, and it was shown by Abelson and diSessa [1], and Hemmerling [79] that in this case a robot using the Pledge algorithm escapes from a polygonal maze, provided that there is such a solution. An example of the robot's path using an error-free Pledge algorithm is given in Figure 3.1. The angle-counting technique is illustrated for the second obstacle: After the robot hits the obstacle in $p_2$ it turns $-\frac{\pi}{2}$ to follow the wall. In $p_3$ the robot turns $+\frac{\pi}{4}$ to follow the next wall. Finally, in $p_4$ it turns $+\frac{\pi}{4}$ again until the angle counter reaches zero and the robot leaves the obstacle. Observe that the robot does not leave the first obstacle in $p_1$, because its angle counter is $-2\pi$ instead of zero.

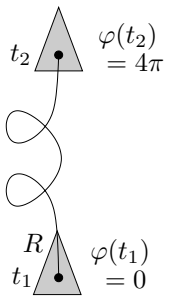
[1] For an implementation of the Pledge algorithm with error-prone angle counting see [78].




Even if the robot is equipped with a compass device, the Pledge algorithm deals only with relative turning angles and does not use the compass' ability to give an absolute direction. However, this is no weak point of the Pledge algorithm, because any strategy has to track the number of full turns, anyway: A strategy that deals only with absolute directions and decides, for example, to leave an obstacle as soon as the compass points to *north* (i. e., $\omega \bmod 2\pi = 0$ holds) can be trapped as shown in the figure. Therefore, a robot equipped with a compass device does not extend the robot model that is assumed for the Pledge algorithm.

### 3.1.2 Sufficient Conditions

Let us assume that it is possible to leave the given maze. Our basic idea is to define a class, $\mathcal{K}$, of curves in the robot's workspace. The curves in $\mathcal{K}$ represent possible paths that lead to an exit, even if the robot's sensors and motions are erroneous. We do not guarantee that $\mathcal{K}$ contains every possible path to an exit, but every curve in $\mathcal{K}$ leads to an exit.



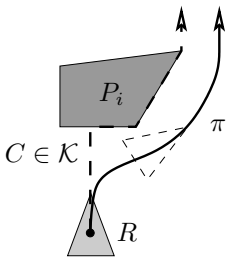

The current location of the robot is given by a *position* and a *heading*. Thus, a curve $C \in \mathcal{K}$ is a subspace of the workspace $\mathcal{C} = \mathbb{R} \times \mathbb{R} \times \mathbb{R}$, and a point on $C$ is described as $C(t) = (P(t), \varphi(t))$, where $P(t) = (X(t), Y(t))$ denotes the position at time $t$ and $\varphi(t)$ the heading. Note that $\varphi(t)$ is the sum of all turns the curve has made so far. For example, after the curve has made two full counterclockwise turns the heading $\varphi(t)$ is equal to $4\pi$ instead of zero.

To classify the possible positions in the workspace, we divide the space of positions, $\mathcal{P} = \mathbb{R} \times \mathbb{R}$, into three subspaces: First, the space of forbidden configurations, $\mathcal{C}_{\text{forb}}$—the union of the interior of all obstacles. Second, the space of half-free configurations, $\mathcal{C}_{\text{semi}}$, that is the union of the boundaries of the obstacles, and finally the free configurations, $\mathcal{C}_{\text{free}}$, where $P(t) \notin P_i$ for all obstacles $P_i$ holds.[2]

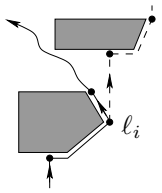
If a curve hits a point $P(h_i)$ in $\mathcal{C}_{\text{semi}}$ after a movement through $\mathcal{C}_{\text{free}}$, we call $h_i$ a *hit point*. If the curve leaves $\mathcal{C}_{\text{semi}}$ and enters the free space at $P(\ell_i)$, we call $\ell_i$ a *leave point*. With respect to the Pledge algorithm we assume that every leave point belongs to a vertex of an obstacle.

[2]To cut short, we use terms like "$t \in \mathcal{C}_{\text{free}}$" instead of "$t$ with $P(t) \in \mathcal{C}_{\text{free}}$".

To escape from an unknown maze, the robot's strategy is not required to move a path that exactly matches a curve in $\mathcal{K}$, rather it is sufficient that the robot's path orients mainly on a curve in $\mathcal{K}$. For example, the robot may follow a wall in a certain distance, because it is not point shaped, see the figure, or it may follow a wall in a zig-zag manner. Therefore, we can describe the parts of a curve that correspond to a movement along a wall as line segments on the boundaries of obstacles.




The Pledge algorithms uses two types of movements: moving along a straight line in the free space, and moving along an obstacle counting the turning angles. Both types of movements may be error prone. Either the turning angles are not measured exactly and the robot leaves the obstacle earlier or later than expected, or the robot cannot follow its initial direction during the movement in the free space.




Thus, we can distinguish between two sources of errors in the Pledge algorithm. Each of them leads to a condition, and both conditions together ensure the correctness of an error-prone Pledge algorithm. In the following we establish a set of sufficient conditions for escaping from the maze. To ensure that the robot can escape following a curve $C \in \mathcal{K}$ we want to avoid infinite cycles in the curve.

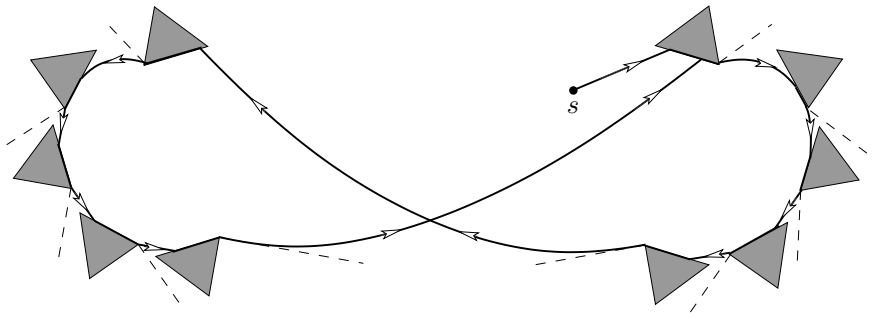



Figure 3.2: Small errors along each boundary can sum up to a cycle.

To establish the set of conditions, we first observe that already small counting errors along the boundary of obstacles or deviations in the free space can sum up to a big mistake and lead to an infinite cycle. Figure 3.2 shows an example; the dashed lines show the correct leaving direction with respect to the direction in the hit point. Between the obstacles the robot drifts a little bit away from the correct direction and ends up in an infinite

loop. This would happen even if the curve would not be allowed to make a full $2\pi$ turn in the free space. Obviously, cycles would be inhibited, if the curve between two obstacles would stay in a wedge around the initial direction. In fact, this wedge can be as large as a half space! This leads to the condition that for any two points in the free space, the difference in the headings should be lower than $\pi$. We refer to this as the *free space condition*.

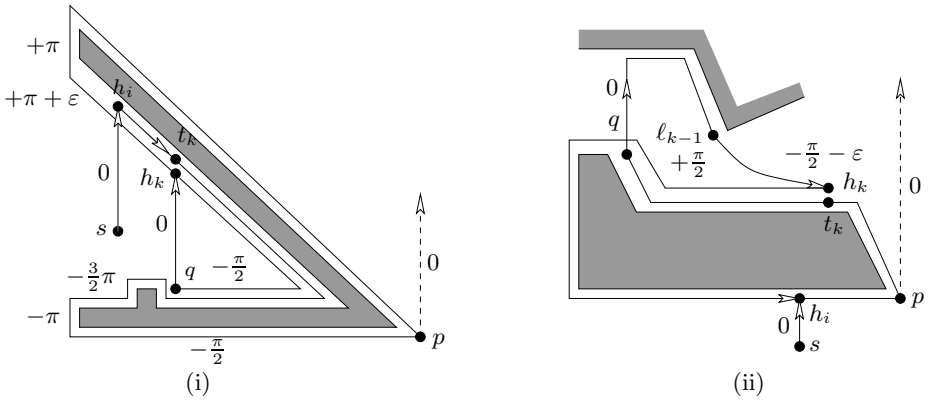



Figure 3.3: Missing a leave point can lead to a cycle.

Unfortunately, the free space condition is not sufficient. Figure 3.3 shows two examples, where $C$ has a cycle, although the free space condition is fulfilled. In both cases, the curve starts in $s$, meets an obstacle in $h_i$, misses the first possible leave point, $p$, and leaves the obstacle at another vertex, $q$. The curve in Figure 3.3(i) hits the same obstacle again in $h_k$. In Figure 3.3(ii) the curve hits another obstacle, leaves this one in $\ell_{k-1}$, and hits the first obstacle again in $h_k$. In both cases, $P(h_k)$ is visited two times, at $h_k$ and $t_k$. In the first case, the heading in $t_k$ is slightly larger than $\pi$, in the second case $+\frac{\pi}{2}$. Observe that the curve in Figure 3.3(ii) represents a second mistake: The heading in the hit point $h_k$ is not zero, as it should be according to the Pledge algorithm. This may occur if the preceding obstacle was left too early or the path between the obstacles is not a straight line segment, or this may be a combination of both reasons. However, the heading in $h_k$ is $-\frac{\pi}{2} - \varepsilon$ and both mistakes sum up to an error that is slightly larger than $\pi$, too.

Note that the problem is not related to a second visit of a single point; the curve of the error-free Pledge algorithm may have many self hits. Instead, the reason is that the heading in $t_k$ is—so to speak—overwinded with respect to the heading in the hit point.

These observations lead to the conjecture that $\varphi(t_k) - \varphi(h_k) < \pi$ should hold whenever the curve hits an obstacle at $h_k$ and there exists a point $t_k$ with $P(h_k) = P(t_k)$. We refer to this as the *obstacle condition*. In the following, we show that both conditions together are sufficient.

**Definition 3.1** Let $\mathcal{K}$ be the class of curves in $C \subseteq \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{semi}}$ that satisfy the following conditions:

(i) The curve $C$ circles an obstacle in a counter-clockwise direction.
(ii) Every leave point belongs to a vertex of an obstacle. Furthermore, for every hit point there is a corresponding leave point, unless the searcher is trapped in a courtyard (see page 79).
(iii) $\forall t_1, t_2 \in C : P(t_1), P(t_2) \in \mathcal{C}_{\text{free}} \Rightarrow |\varphi(t_1) - \varphi(t_2)| < \pi$ (free space condition)
(iv) $\forall h_i, t \in C : P(t) = P(h_i) \Rightarrow \varphi(t) - \varphi(h_i) < \pi$ (obstacle condition).

Obviously, the curve of the error-free Pledge algorithm is a curve in $\mathcal{K}$. The curves in $\mathcal{K}$ have two important properties that we show in the following two lemmata.

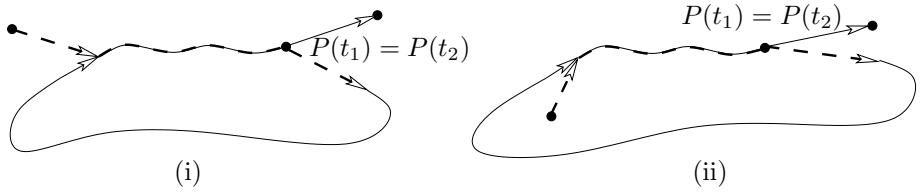



Figure 3.4: The difference between (i) a crossing and (ii) a touch at $t_2$.

**Lemma 3.2** *A curve $C \in \mathcal{K}$ cannot cross itself.*

Note that a curve of $\mathcal{K}$ can touch itself, see Figure 3.4.

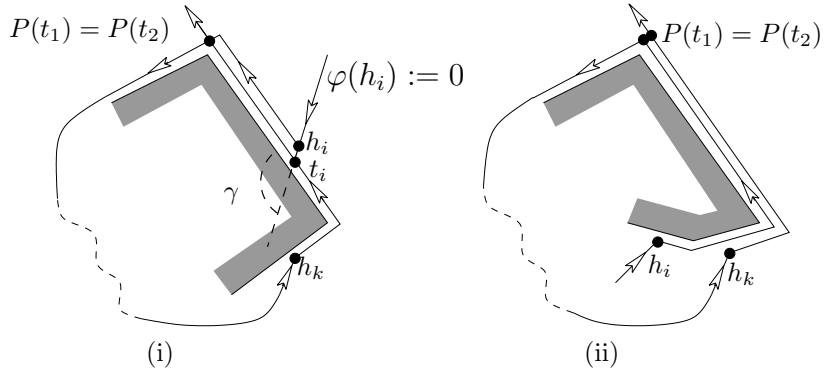



Figure 3.5: (i) A counterclockwise turn and a crossing, (ii) no crossing.

**Proof.** Let us assume, $C$ crosses itself. Consider the *first* crossing of $C$; that is, there are two parameters, $t_1$ and $t_2$, with $t_1 < t_2$ and $P(t_1) = P(t_2)$, so that a crossing occurs in $P(t_2)$ and no crossing exists before $t_2$. The curve $C$ makes either a counterclockwise or a clockwise loop between $t_1$ and

$t_2$. If a crossing happens in the free space, the curve violates the free space condition; thus, we assume that the crossing occurs in $\mathcal{C}_{\text{semi}}$.

Let us consider the case of a counterclockwise loop, see Figure 3.5(i). The curve hits an obstacle at $h_i$, makes a counterclockwise turn, meets $P(h_i)$ at $t_i$ again, and has a crossing at $t_2 > t_i > h_i$. Note that there is no crossing, if the point $P(h_i)$ is not met between $t_1$ and $t_2$, see Figure 3.5(ii).

W. l. o. g. we assume $\varphi(h_i) = 0$. The loop may leave the obstacle or not; however, we reach $t_i$ with the heading $\varphi(t_i) + (-\gamma) = 2\pi$. At $P(h_i)$ the robot turns clockwise with angle $\gamma$ to follow the obstacle boundary, so $-\pi < \gamma < 0$ must hold. Hence, $\varphi(t_i)$ is greater than $\pi$ and the obstacle condition is violated.

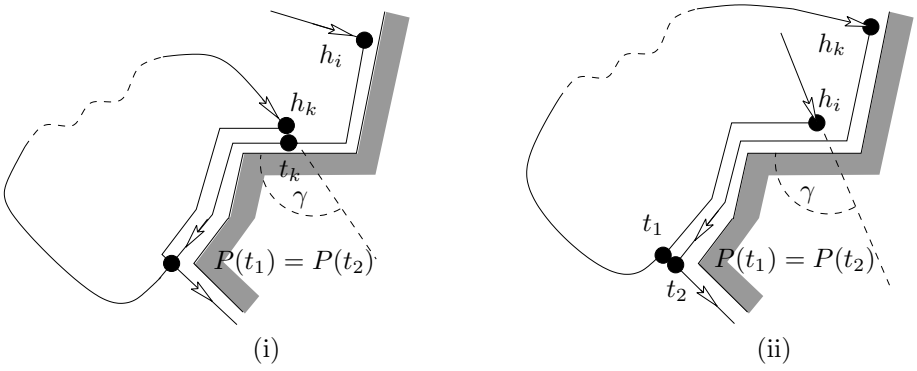



Figure 3.6: (i) A clockwise turn and a crossing, (ii) no crossing.

Now we look at a clockwise turn. The curve hits an obstacle at $h_i$, follows the obstacle and leaves the obstacle. Eventually, it returns to the obstacle at another hit point $h_k > h_i$ and has a crossing at $t_2$, see Figure 3.6(i). The point $P(h_k)$ has to be met before at $t_k$ between $h_i$ and $t_1$. Otherwise, the curve only touches itself and there is no crossing at $t_2$, see Figure 3.6(ii).

Let $\varphi(h_k^+)$ denote the heading immediately after the robot has turned in $h_k$; that is, $\varphi(h_k^+) = \varphi(h_k) + \gamma$. Again, we have $-\pi < \gamma < 0$. On the other hand, the curve has made a full clockwise turn between $t_k$ and $h_k^+$; thus, $\varphi(h_k^+) = \varphi(t_k) - 2\pi$. Finally, the obstacle condition has to be fulfilled, too:

$$\begin{aligned} & \varphi(t_k) - \varphi(h_k) < \pi \\ \Leftrightarrow\quad & \varphi(h_k^+) + 2\pi - \varphi(h_k) = \varphi(h_k) + \gamma + 2\pi - \varphi(h_k) < \pi \\ \Leftrightarrow\quad & \gamma < -\pi \quad \lightning \end{aligned}$$

It follows that the first crossing cannot exist, and—by induction—the curve cannot cross itself. □

**Lemma 3.3** *A curve $C \in \mathcal{K}$ hits every edge in the environment at most once.*

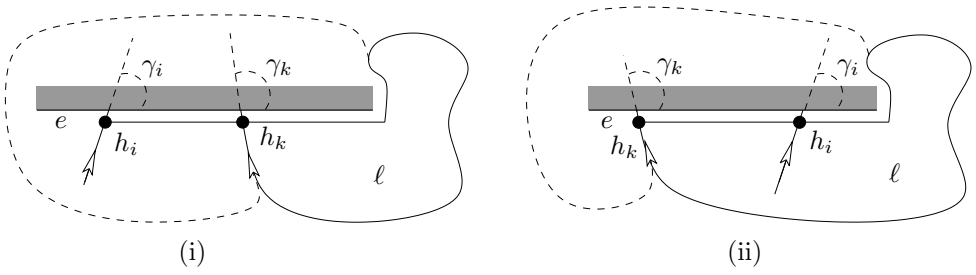



Figure 3.7: A curve that hits an edge twice.

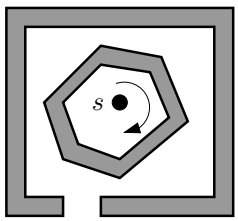
**Proof.** Let us assume, the curve $C$ hits an edge $e$ more than once: After a first hit at $h_i$ the robot moves around and hits $e$ again at $h_k$, see Figure 3.7. In $P(h_i)$ and $P(h_k)$ the robot turns clockwise to follow the edge $e$, therefore $-\pi < \gamma_i, \gamma_k < 0$ holds. Let $\varphi(h_i^+)$ and $\varphi(h_k^+)$ be defined as in the proof of Lemma 3.2. W. l. o. g. we assume $\varphi(h_i^+) = 0$. Because the curve in $h_i^+$ and $h_k^+$ follows the same edge $e$, the headings $\varphi(h_i^+)$ and $\varphi(h_k^+)$ must be mod $2\pi$ the same; thus, $\varphi(h_k^+) = 2j\pi, j \in \mathbb{Z}$. For $j \neq 0$ follows with $\varphi(h_i) = -\gamma_i$ and $\varphi(h_k) = \varphi(h_k^+) - \gamma_k$ that $|\varphi(h_k) - \varphi(h_i)| = |2j\pi - \gamma_k + \gamma_i| > \pi$ holds; thus, the free space condition is violated.

Therefore, we can assume $j = 0$ and $\varphi(h_k^+) = 0$. Consider the part of $C$ between the first and the second visit of $P(h_k)$ (in a situation as shown in Figure 3.7(i)), or the curve between the consecutive visits of $P(h_i)$ as shown in Figure 3.7(ii). If this loop, $\ell$, has no crossings, then $\ell$ is a Jordan curve and $C$ makes a $\pm 2\pi$ turn in $\ell$. Thus, $\varphi(h_k^+)$ is equal to $\pm 2\pi$, in contradiction to our assumption. Hence, the curve between the two visits of $e$ must have at least one crossing. But this contradicts to Lemma 3.2. □

Finally, with Lemma 3.2 and Lemma 3.3 we are able to show that the conditions from Definition 3.1 are sufficient to solve our problem.

**Theorem 3.4** *A robot whose path follows a curve $C \in \mathcal{K}$ escapes from an unknown maze, if this is possible at all.*

**Proof.**




By Lemma 3.3, a curve that meets the conditions from Definition 3.1 hits every edge in the environment at most once. When every edge is visited, the robot either escapes from the next leave point, because it cannot hit any further edge, or it is not able to leave the currently visited obstacle at all. But by Definition 3.1(ii) every obstacle will be left, unless the searcher is trapped in a courtyard, see the figure. Thus, if the robot never leaves the currently visited obstacle, it is not possible to escape from the maze. □

### 3.1.3 Applications

In this section we consider the practical relevance of Theorem 3.4. What consequences does it have for the design of a robot that should be able to leave an unknown maze?

#### 3.1.3.1 Leaving a Maze Using an Error-Prone Compass

Now, let us return to the compass-equipped robot. Theorem 3.4 can be easily applied to determine the maximal error in the compass readings.

**Corollary 3.5** *A robot equipped with a compass device finds the exit from an unknown maze using a simple Pledge-like strategy, if the error in the compass readings is smaller than $\frac{\pi}{2}$; provided that it is possible to leave the maze.*

**Proof.** If the compass has a measuring error less than $\frac{\pi}{2}$, it is easy to ensure that the heading of the robot in the free space remains in the interval $]-\frac{\pi}{2}, \frac{\pi}{2}[$. Thus, the free space condition is satisfied. Additionally, at every detected hit point $h_i$ we have $\varphi(h_i) \in ]-\frac{\pi}{2}, \frac{\pi}{2}[$.

Further, it is easy to detect $+2\pi$ or $-2\pi$ turns along the walls within a deviation of $]-\frac{\pi}{2}, \frac{\pi}{2}[$ due to the compass inaccuracy. Therefore, we can assume that the deviation between the measured turning angle on a path along the walls and the actual turning angle is less than $\frac{\pi}{2}$, so $\varphi(t) < \frac{\pi}{2}$ is always satisfied while the robot follows a wall. Altogether, the obstacle condition is fulfilled. □

#### 3.1.3.2 Exact Free Motion

Let us assume that the robot is able to move along a straight path between obstacles correctly, or the deviations on a straight path are neglectable; that is, the robot always hits the same edge as an error-free robot using the Pledge algorithm. Only the angle counter of the robot is inaccurate in some way. Let $\beta_i$ denote the difference between the real angle (the nominal value) and the measured angle at the $i$th turn, and $n$ the number of vertices in the environment.

**Lemma 3.6** *If the robot is able to move along a straight path in the free space and ensures that $\left|\sum_{i=k}^{\ell} \beta_i\right| < \pi$ holds for all $k \leq \ell \leq m$, where $m$ denotes the number of turns the robot has made so far, then it is able to escape from an unknown maze using the Pledge algorithm.*

**Proof.** The new condition states that the absolute value of the accumulated measuring error—the difference between the robot's heading and its angle

counter—never exceeds $\pi$. Now, we have to show that our new condition does not violate the conditions from Definition 3.1.

First, let us assume that the free space condition is not met, so there must be two points $t_1, t_2$ in the free space where $|\varphi(t_1) - \varphi(t_2)| \geq \pi$ holds. W. l. o. g. we assume $\varphi(t_1) = 0$. Because the robot correctly moves along a straight line in the free space, the headings at the leave point, the following hit point, and all points between them must be the same. Thus, there must be a leave point $\ell_k$ with $|\varphi(\ell_k)| = |\varphi(t_2)| \geq \pi$. But the robot leaves an obstacle only, if its angle counter has reached zero, so the absolute value of the accumulated measuring error in $\ell_k$ is at least $\pi$.

Second, we assume that the obstacle condition is violated. Then there must be an obstacle $P_i$ with a hit point $h_k$ and another point $t_k$ with $P(t_k) = P(h_k)$, such that $\varphi(t_k) - \varphi(h_k) > \pi$ holds. W.l.o.g., let $\varphi(h_k) = 0$, then $\varphi(t_k) \geq \pi$ holds, but the angle counter cannot be greater than zero, because the robot leaves the obstacle as soon as the counter becomes zero. Thus, the difference between the robot's heading and its angle counter must be at least $\pi$. □

If the conditions from Definition 3.1 are satisfied and the robot escapes, the robot visits every vertex at least once between two consecutive hit points. Thus, with Lemma 3.3 the robots visits at most $n^2$ vertices. Now, let the robot's maximal error be $\beta_{\max} := \max \beta_i$. With Lemma 3.6 we have:

**Corollary 3.7** *A robot that guarantees $|\beta_{\max}| < \frac{\pi}{n^2}$ is able to escape from an unknown maze with the Pledge algorithm; provided that it is possible to leave the maze.*

#### 3.1.3.3 (Pseudo-) Orthogonal Scenes

In this section, we observe the Pledge algorithm in a special case of scenes—orthogonal scenes—, but allow them to be inaccurate. A scene is called *orthogonal*, if every polygon in the scene is orthogonal; that is, the polygon edges meet with internal angles of either $\frac{\pi}{2}$ or $\frac{3}{2}\pi$ [150].[3]

W. l. o. g. we assume that the polygon edges are axis parallel. For orthogonal polygons we call a vertex *convex*, if its inner angle is equal to $\frac{\pi}{2}$, and *reflex*, if its inner angle is $\frac{3}{2}\pi$.

It is easy to see that we can simplify the Pledge algorithm in orthogonal scenes. Because we have only two types of vertices, it is sufficient to keep track of the number of convex and reflex vertices that the robot passes by. Thus, the angle counting amounts to count $+1$ for a reflex vertex and $-1$ for a convex vertex. Additionally, we count $+1$ for the hit point, see Figure 3.8(i).

Now, we assume that the environment is more or less orthogonal; we allow small deviations from the strict axis parallelism. To quantify these

[3] Polygons with this property are called also *rectilinear*.

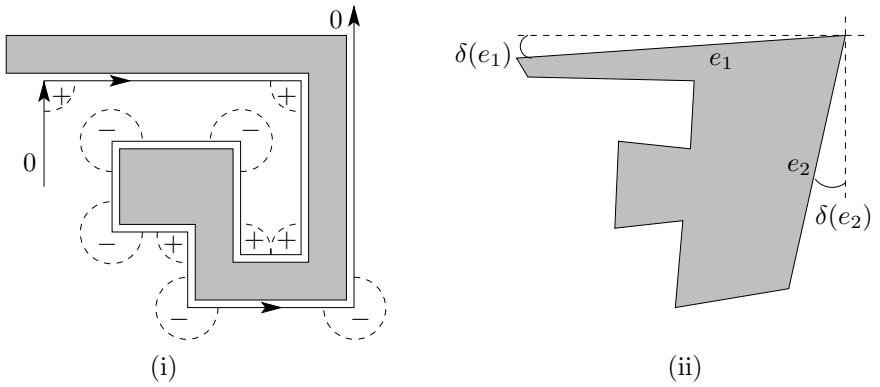



Figure 3.8: (i) Angle counting for an orthogonal polygon, (ii) pseudo-orthogonal polygon and divergence $\delta$.

deviations, we use the angle of edges in a tilted position. More precisely, we define the *divergence*, $\delta(e)$, of an edge $e = (v, w)$ as the smallest angle between $e$ and an axis-parallel line through $v$ or $w$, see Figure 3.8(ii).

**Definition 3.8** A simple polygon, $P$, is called *pseudo orthogonal*, if

$$\#\text{convex vertices of } P \;=\; \#\text{reflex vertices of } P + 4$$

holds.[4] We call a pseudo-orthogonal polygon, $P$, *$\delta$-pseudo orthogonal*, if $P$ satisfies

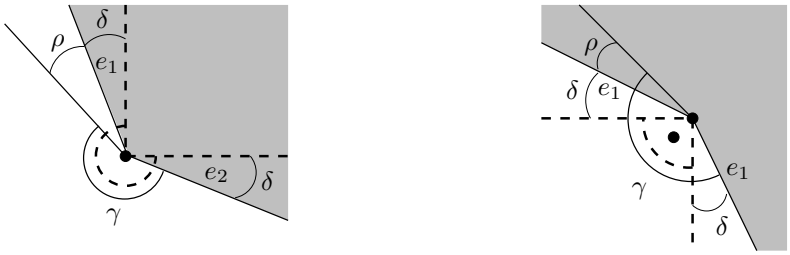
$$\delta(P) := \max_{e \in P} \delta(e) \leq \delta\,.$$

A set of polygons, $\mathcal{P}$, is called ($\delta$-)pseudo-orthogonal scene, if every $P_i \in \mathcal{P}$ is ($\delta$-)pseudo orthogonal.

The robot's angle counter may be a second source of errors. We assume that the robot is able to measure angles with an accuracy of $\rho$; that is, the difference between the nominal value and the measured value is smaller than $\rho$. Now, we are interested in upper bounds for $\delta$ and $\rho$ that guarantee a successful application of the simplified Pledge algorithm to a $\delta$-pseudo-orthogonal scene.

First, we have to guarantee that the robot is able to distinguish convex and reflex vertices correctly. Taking the worst case into account, we assume that both edges adjacent to the current vertex deviate with the maximal angle $\delta$ and the robot's measuring error is maximal, too, see Figure 3.9. To ensure a correct classification, we require that the measured outer angle, $\gamma$, is greater than $\pi$ for a convex vertex, and smaller than $\pi$ for a reflex vertex.

[4] A *convex* (*reflex*) vertex of a simple polygon is a vertex with an inner angle smaller (greater) than $\pi$.



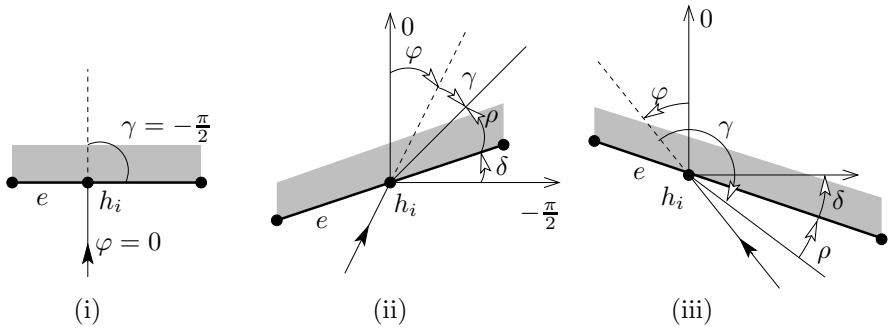

convex vertex reflex vertex

Figure 3.9: Maximal deviation between the outer measured angle ($\gamma$) and the outer nominal value (dashed) for a convex and a reflex vertex.

Thus, we have to ensure

$$\frac{3}{2}\pi - 2\delta - \rho > \pi \quad \Leftrightarrow 2\delta + \rho < \frac{\pi}{2} \qquad \text{and} \qquad \frac{\pi}{2} + 2\delta + \rho < \pi \quad \Leftrightarrow 2\delta + \rho < \frac{\pi}{2}\,.$$

On the other hand, the robot should be able to maintain its initial direction, $\omega$, in the free space, see Algorithm 3.1. To establish bounds for the maximal deviation from the initial direction, we consider the headings in the hit points. In the error-free case, the robot hits only horizontal edges, but in a $\delta$-pseudo-orthogonal scene the robot has to determine whether an edge is classified as *horizontal* or as *vertical*. If the robot hits a *horizontal* edge, it has to surround an obstacle using the simplified angle-counting procedure, whereas a *vertical* edge can be ignored; that is, the robot just slides along this edge.




(i) (ii) (iii)

Figure 3.10: The robot hits a horizontal edge (i) error-free case, (ii) small absolute value for $\gamma$, (iii) large absolute value for $\gamma$.

As in the preceding case, let $\gamma$ denote the *measured* angle that the robot turns in a hit point to follow a wall. Considering the divergence in the edges, it is reasonable that the robot assumes an edge to be *horizontal*, if $-\frac{1}{4}\pi > \gamma > -\frac{3}{4}\pi$ holds, and *vertical* otherwise. Figure 3.10(ii) and (iii)

show the worst cases for $\gamma$. The robot hits an edge $e$ in $h_i$ with the heading $\varphi = \varphi(h_i)$. In (ii) the deviations $\varphi, \delta$ and $\rho$ make the absolute value of $\gamma$ as small as possible, in (iii) as large as possible. To ensure that $\gamma \in \,]-\frac{1}{4}\pi, -\frac{3}{4}\pi\,[$ holds, we have to restrict the heading, $\varphi$, in the hit point.
From Figure 3.10(ii) we get

$$\gamma = -\frac{\pi}{2} - \varphi + \delta + \rho < -\frac{\pi}{4} \;\Leftrightarrow\; -\frac{\pi}{4} + \delta + \rho < \varphi\,,$$

and Figure 3.10(iii) yields

$$\gamma = -\Big(\frac{\pi}{2} + \varphi + \delta + \rho\Big) > -\frac{3}{4}\pi \;\Leftrightarrow\; \frac{\pi}{4} - \delta - \rho > \varphi\,.$$

Thus, the robot detects a *horizontal* edge correctly, if $\varphi(h_i) \in \,]\, -\frac{\pi}{4} + \delta + \rho, \frac{\pi}{4} - \delta - \rho\,[$ holds, and thus we require $\delta + \rho < \frac{\pi}{4}$, which includes the preceding condition $2\delta + \rho < \frac{\pi}{2}$.

In an orthogonal scene, the robot can rely on the vertical edges to adjust its initial direction. In our case, the robot leaves an obstacle moving along a *vertical* edge, too, but this means that the heading in a leave point is in the range $[-\delta, +\delta\,]$. Considering the range for the heading in the (next) hit point, we conclude that the deviation in the free space with respect to the heading in the leave point has to be smaller than $\frac{\pi}{4} - 2\delta - \rho$. Altogether, we have

**Corollary 3.9** *Let an unknown $\delta$-pseudo-orthogonal scene be given, and let us assume that it is possible to escape from this maze. If the robot is able to measure angles within an accuracy of $\rho$ with $\delta + \rho < \frac{\pi}{4}$, and—in the free space—its deviation from the heading in the preceding leave point is smaller than $\frac{\pi}{4} - 2\delta - \rho$, the robot is able to escape using the Pledge algorithm with the simplified angle counting.*

## 3.2 Finding a Door

As the second application of error-prone robot models we consider the problem of finding a door along a wall with a blind robot, which does not know the location of the door—neither the distance nor the direction towards the door. This problem was considered by Gal [64, 65, 6] and independently reconsidered by Baeza-Yates et al. [11]. Both works led to the *doubling strategy*, which is a basic paradigm for search algorithms; for example, searching for a point in a polygon, see Klein [115] and Schuierer [161], or approximating the optimal search path, see Chapter 4.

Searching on the line was generalized to searching on $m$ rays emanating from a single source, see [64, 11]—this is also known as the *lost cow problem*. Many other variants were discussed since then, for example $m$-ray searching with restricted goal distance (Hipke et al. [82], Langetepe [122], López-Ortiz and Schuierer [162, 131]), $m$-ray searching with additional turn costs (Demaine et al. [41]), parallel $m$-ray searching (Kao et al. [108], Hammar et al. [76], López-Ortiz and Schuierer [133]) or randomized searching (Schuierer [163], Kao et al. [109]).

We investigate the impact of an error in the movement to the correctness and the competitive factor of a strategy. In the first setting, the error range, denoted by a parameter $\delta$, is not known to the strategy. The strategy acts as in the error free case. Afterward, we consider the case that the strategy is aware of the error range and takes the error into account.

### 3.2.1 The Doubling Strategy

The task is to find a door in a wall, or rather a point, $t$, on a line. The robot does not know whether $t$ is located left hand or right hand to its start position, $s$, nor does it know the distance from $s$ to $t$. We assume that the distance to the door is at least 1; thus, the additive constant in the definition of the competitive factor—see Definition 1.1—can be omitted, see, for example, Langetepe [122].

We can describe a strategy to solve this problem using a sequence $F = (f_i)_{i \in \mathbb{N}}$. $f_i$ denotes the search depth, that is the distance that the robot walks in the $i$th step. If $i$ is even, the robot moves $f_i$ units from the start to the right and $f_i$ units back to the left; if $i$ is odd, the robot moves to the left and back to the right. Gal and Baeza-Yates et al. showed that the strategy $f_i = 2^i$ is 9-competitive, assuming that the movement is correct; that is, after moving $f_i$ units from the start point to the right and moving $f_i$ units to the left—and vice versa—the robot has reached its start point. Further, they showed that no other strategy can achieve a competitive factor smaller than 9.

### 3.2.2 Modeling the Error

The robot moves straight line segments of a certain length from the start point alternately to the left and to the right. Every movement can be erroneous, which causes the robot to move more or less far than expected. However, we require the robots error per unit to be within a certain error bound, $\delta$. More precisely, let $f$ denote the length of a movement required by the strategy—the nominal value—and let $\ell$ denote the actually covered distance, then we require that $\ell \in [\,(1-\delta)f, (1+\delta)f\,]$ holds for $\delta \in [\,0, 1\,[$; that is, the robot moves at least $(1-\delta)f$ and at most $(1+\delta)f$. This is a reasonable error model, because the actually covered distance is in a symmetrical range around the nominal value. Another commonly used method is to require $\ell \in [\,\frac{1}{1+\delta'}f, (1+\delta')f\,]$ for $\delta' > 0$. This leads to an unsymmetrical range around the nominal value, but does not restrict the upper bound for the error range. Because both error models may be of practical interest, we give results for both models.

### 3.2.3 Disregarding the Error

In this section, we assume that the robot is not aware of making any errors. In this case, the optimal doubling strategy presented in Section 3.2.1 seems to be the best choice for the robot. Now, the question is whether the robot is still able to reach the door, or rather, whether there is an upper bound for the error, $\delta$, that still guarantees a success. Further, we want to analyze the worst-case efficiency for a doubling strategy with respect to $\delta$. We are also interested in the number of additional iteration steps that an error-prone robot needs compared to the error-free case.

W. l. o. g. we consider the case that the door is located right hand to the start point, $s$. The other case is handled analogously.

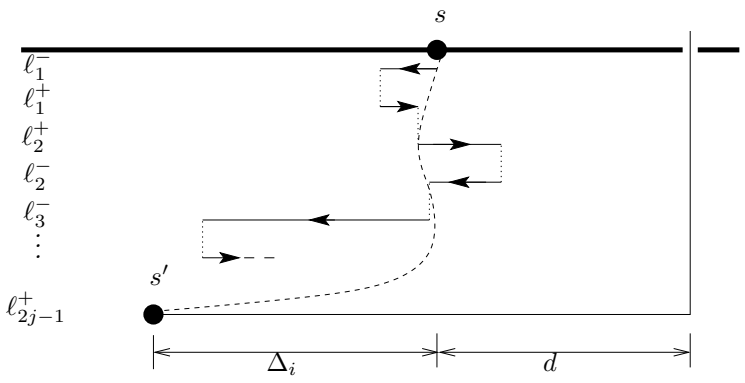



Figure 3.11: The $i$th iteration consists of two separate movements, $\ell_i^+$ and $\ell_i^-$. Both may be of different length, causing a drift. The vertical path segments are to highlight the single iterations, the robot moves only on horizontal segments.

The errors in the movements away from the door and back towards the door may be different, so the robot may not return to the start point, $s$, between two iterations, see Figure 3.11. Even worse, the start point of every iteration may drift continuously away from the original start point. Let $\ell_i^+$ be the length of the movement to the right in the $i$th step and $\ell_i^-$ be the covered distance to the left. Now, the deviation from the start point after the $k$th iteration step, the *drift* $\Delta_k$, is

$$\Delta_k = \sum_{i=1}^{k} (\ell_i^- - \ell_i^+).$$

If the drift is greater than zero, the start point $s_{k+1}$ of the iteration $k+1$ is located left to the original start point, if it is smaller than zero, $s_{k+1}$ is right hand to $s$. Note that $\ell_i^+$ is equal to $\ell_i^-$ in the error-free case.
The length of the path $\pi_k$ after $k$ iterations is

$$|\pi_k| = \sum_{i=1}^{k} (\ell_i^- + \ell_i^+).$$

#### 3.2.3.1 Reachability

The first question is, which condition must hold to guarantee that the robot reaches the door. Further, we are interested in the number of additional iteration steps that the error-prone robot needs compared to the error-free robot, depending on its maximal error bound, $\delta$. We assume that the door is located at $d = 2^{2j} - \varepsilon$, so an error-free robot hits the door in the iteration step $2j$ with a search depth of $f_{2j} = 2^{2j}$. The faulty robot may miss the door during the $2j$th iteration due to the drift to the left, but hits the door in a subsequent iteration step $2j + 2k$, $k \in \mathbb{N}_0$.

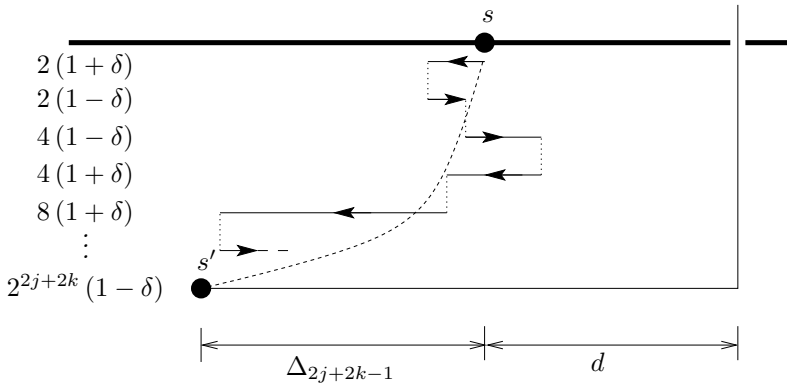



Figure 3.12: In the worst case, the start point of every iteration drifts away from the door.

To guarantee reachability, we have to assume that the drift away from to door is maximal; that is, every step of the error-prone robot towards the

door is too short and every step to the other side is too long, see Figure 3.12. Thus, we have $\Delta_k = 2\delta\,(2^{k+1} - 2)$. To ensure that the robot hits the door, the covered distance in the final step has to be at least as large as the sum of the distance to the door, $d$, and the overall drift to the left, $\Delta_{2j+2k-1}$. The final straight path may be erroneous again, but its length is a least the lower bound of the covered distance in the iteration $2j + 2k$, that is $(1-\delta)\,2^{2j+2k}$. Altogether we have

$$\begin{aligned}
& \Delta_{2j+2k-1} + d \le (1-\delta)\,2^{2j+2k} \\
\Leftrightarrow\quad & 2\delta \cdot 2^{2j+2k} - 4\delta + 2^{2j} - \varepsilon \le (1-\delta)\,2^{2j+2k} \\
\Leftrightarrow\quad & 2\delta + \frac{1}{2^{2k}} - \frac{\varepsilon + 4\delta}{2^{2j+2k}} \le 1 - \delta \\
\Leftrightarrow\quad & \delta \le \frac{1}{3} \cdot \frac{2^{2k} - 1}{2^{2k}} + \frac{\varepsilon + 4\delta}{3 \cdot 2^{2j+2k}} \\
\Leftarrow\quad & \delta \le \frac{1}{3} \cdot \frac{2^{2k} - 1}{2^{2k}}
\end{aligned}$$

If we allow an arbitrary number of additional iterations, the latter term converges for $k \to \infty$ to $\frac{1}{3}$ and we get

**Corollary 3.10** *Let the distance covered by the robot, $\ell$, be in the range $[\,(1-\delta)f, (1+\delta)f\,]$ for $\delta \in [\,0, 1\,[$.*
*If the error $\delta$ is not greater than $\frac{1}{3}$ the robot reaches the door; that is, for every $d = 2^{2j} - \varepsilon$ we can find a $k \in \mathbb{N}_0$ such that the robot hits the door in the iteration step $2j + 2k$.*

We can also relate the number of additional iterations to the error bound. For example, we can set $k$ equal to one and get:

**Corollary 3.11** *If the error $\delta$ is not greater than $\frac{1}{4}$ in the error model $\ell \in [\,(1-\delta)f, (1+\delta)f\,]$, the robot reaches the door after at most one iteration step towards the door more than the error-free robot; that is, for every $d = 2^{2j} - \varepsilon$ the robot hits the door at least in the iteration step $2j + 2$.*

Remark that both bounds are tight; that is, if $\delta > \frac{1}{3}$ ($\delta > \frac{1}{4}$) holds, we can find a distance $d = 2^{2j} - \varepsilon$, such that the robot does not reach the goal with an arbitrary number of steps ($2j + 2$ steps, respectively).

In the second error model we have

$$\begin{aligned}
& \Delta_{2j+2k-1} + d \le \frac{1}{1+\delta}\,2^{2j+2k} \\
\Leftrightarrow\quad & \left(1 + \delta - \frac{1}{1+\delta}\right)\left(2^{2j+2k} - 2\right) + 2^{2j} - \varepsilon \le \frac{1}{1+\delta}\,2^{2j+2k} \\
\Leftrightarrow\quad & \left((1+\delta)^2 - 1\right)\left(1 - \frac{2}{2^{2j+2k}}\right) + \frac{1+\delta}{2^{2k}} - \frac{\varepsilon\,(1+\delta)}{2^{2j+2k}} \le 1
\end{aligned}$$

$$\Leftarrow \quad \delta^2 + \left(2 + \frac{1}{2^{2k}}\right)\delta + \frac{1}{2^{2k}} - 1 \leq 0$$
$$\Leftrightarrow \quad \delta \leq \sqrt{2 + \frac{1}{2^{4k+2}}} - 1 - \frac{1}{2^{2k+1}}$$

This yields, for example, for $k \to \infty$ and $k = 1$:

**Corollary 3.12** *If the error $\delta$ is not greater than $\sqrt{2}-1$ in the error model $\ell \in [\,\frac{1}{1+\delta}f, (1+\delta)f\,]$, the robot reaches the door with an arbitrary number of additional steps, and for $\delta \leq \sqrt{\frac{129}{64}} - \frac{9}{8} \approx 0.29$ the robot hits the door after only two iteration steps more than the error-free robot.*

#### 3.2.3.2 Competitive Factor

Now, we analyze the performance of the doubling strategy $f_i = 2^i$ with an error-prone robot in the competitive framework.

**Theorem 3.13** *Let the distance covered by the robot, $\ell$, be in the range $[\,(1-\delta)f, (1+\delta)f\,]$ for $\delta \in [\,0, 1\,[$. If the error $\delta$ is not greater than $\frac{1}{3}$ the robot finds the door with the doubling strategy $f_i = 2^i$. The covered path is not longer than $1 + 8\,\frac{1+\delta}{1-3\delta}$ times the shortest path to the door.*[5]

**Proof.** For the competitive setting it is the worst, if the door is hit in the iteration step $2j+2$, but located just a little bit farther away than the rightmost point that was reached in the $2j$th iteration. Additionally, we have to consider the case, in which the goal is exactly one unit away from the start. We discuss this case at the end of this proof.

We want the door to be located closely behind the rightmost point visited in the iteration step $2j$. Considering the drift $\Delta_{2j-1}$, the distance from the start point $s$ to the door is

$$d = \ell^+_{2j} - \sum_{i=1}^{2j-1} (\ell^-_i - \ell^+_i) + \varepsilon\,.$$

The total path length is the sum of the covered distances up to the start point of the final iteration, the distance from this point to the original start point $s$ (i. e., the overall drift), and the distance to the door:

$$|\pi_{\text{onl}}| = \sum_{i=1}^{2j+1} (\ell^-_i + \ell^+_i) + \sum_{i=1}^{2j+1} (\ell^-_i - \ell^+_i) + d\,.$$

[5] More precisely, the factor is $1 + 8\frac{1+\delta}{1-3\delta+\varepsilon}$ for an arbitrary small $\varepsilon$, which is crucial for the case $\delta = \frac{1}{3}$, but neglectable in all other cases. For convenience we omit the $\varepsilon$ in this and the following theorems.

Thus, we have the worst-case ratio

$$\frac{|\pi_{\text{onl}}|}{d} \leq 1 + \frac{\sum_{i=1}^{2j+1} 2\ell_i^-}{\ell_{2j}^+ - \sum_{i=1}^{2j-1}(\ell_i^- - \ell_i^+) + \varepsilon}\,. \tag{3.1}$$

We can see that this ratio achieves its maximum if we maximize every $\ell_i^-$; that is, if we set $\ell_i^-$ to $(1+\delta)2^i$ in this error model. Now, we have to fix $\ell_i^+$ to maximize the ratio. Obviously, the denominator gets its smallest value if every $\ell_i^+$ is as small as possible, therefore we set $\ell_i^+$ to $(1-\delta)2^i$. Thus, the worst case is achieved—as in the previous section—if every step to the right is too short and every step to left is too long, yielding a maximal drift away from the door, see Figure 3.12. Altogether, we get

$$\begin{aligned}
\frac{|\pi_{\text{onl}}|}{d} &\leq 1 + \frac{\sum_{i=1}^{2j+1} 2\ell_i^-}{\ell_{2j}^+ - \sum_{i=1}^{2j-1}(\ell_i^- - \ell_i^+) + \varepsilon} \\
&= 1 + \frac{2\,(1+\delta)\,\sum_{i=1}^{2j+1} 2^i}{(1-\delta)\,2^{2j} - 2\delta\,\sum_{i=1}^{2j-1} 2^i + \varepsilon} \\
&= 1 + \frac{2\,(1+\delta)\,(2^{2j+2}-2)}{(1-3\delta)\,2^{2j} + 4\delta + \varepsilon} \\
&< 1 + 8\,\frac{1+\delta}{1-3\delta}\,.
\end{aligned}$$

For the case that the goal is exactly one step away from the start, we achieve a factor of $1 + 4\,\frac{1+\delta}{1}$, which is smaller than the preceding factor.

To reach the door, we have to ensure that the rightmost visited point proceeds in every iteration step to the right towards the door; that is, the strategy is strictly monotonically increasing. Thus, the covered distance in the $k$th iteration has to exceed the overall drift:

$$(1-\delta)\,2^k > \Delta_{k-1} \tag{3.2}$$

With the preceding equations for $\Delta_i, \ell_i^+$ and $\ell_i^-$ we have

$$\begin{aligned}
(1-\delta)\,2^k &> 2\delta\,(2^k - 2) \\
\Leftrightarrow (1-3\delta)\,2^k + 4\delta &> 0
\end{aligned}$$

Thus, if $\delta$ is greater than $\frac{1}{3}$, the robot does not pass over the point $4\delta$ right to $s$ in the worst case. If the door is farther to the right, it may not be reached. This corresponds to Corollary 3.10 and to the observation that the denominator of the competitive factor gets negative for $\delta > \frac{1}{3}$. □

**Corollary 3.14** *Let $\ell \in [\,\frac{1}{(1+\delta)}f, (1+\delta)f\,]$ for $\delta > 0$ hold. If $\delta$ is not greater than $\sqrt{2}-1$ the robot reaches the door using the doubling strategy $f_i = 2^i$ with a competitive factor of $1 + 8\frac{(1+\delta)^2}{2-(1+\delta)^2}$.*

**Proof.** In this error model, we set $\ell_i^-$ to $2^i(1+\delta)$ and $\ell_i^+$ to $\frac{2^i}{1+\delta}$. Equation 3.1 yields

$$\begin{aligned}
\frac{|\pi_{\text{onl}}|}{d} &\leq 1 + \frac{2\sum_{i=1}^{2j+1}(1+\delta)\,2^i}{\frac{1}{1+\delta}\,2^{2j} - \sum_{i=1}^{2j-1}\left(1+\delta-\frac{1}{1+\delta}\right)2^i + \varepsilon} \\
&= 1 + \frac{2\,(1+\delta)^2\,(4-\frac{2}{2^{2j}})}{1-((1+\delta)^2-1)\left(1-\frac{2}{2^{2j}}\right)+\frac{\varepsilon\,(1+\delta)}{2^{2j}}} \\
&= 1 + \frac{2\,(1+\delta)^2\,(4-\frac{2}{2^{2j}})}{2-(1+\delta)^2+\left(\frac{2}{2^{2j}}((1+\delta)^2-1)\right)+\frac{\varepsilon\,(1+\delta)}{2^{2j}}} \\
&< 1 + 8\cdot\frac{(1+\delta)^2}{2-(1+\delta)^2}
\end{aligned}$$

Equation 3.2 reads

$$\begin{aligned}
&\frac{1}{1+\delta}\,2^{2j} > (1+\delta-\frac{1}{1+\delta})\,(2^{2j}-2) \\
\Leftrightarrow\ & ((1+\delta)^2-1)\,(2^{2j}-2) < 2^{2j} \\
\Leftrightarrow\ & (1+\delta)^2 < 2+\frac{2}{2^{2j}}((1+\delta)^2-1) \\
\Leftarrow\ & (1+\delta)^2 < 2 \\
\Leftrightarrow\ & \delta < \sqrt{2}-1
\end{aligned}$$

□

So far, we assumed that the robot is not able to recognize the start point $s$ when passing it. One might ask what happens, if the robot is able to detect the start point. In this setting, the we obviously have no drift; that is, $\Delta_i = 0$ and $\ell_i^+ = \ell_i^- =: \ell_i$ holds for every $i$. Indeed, this is just a special case of the $m$-ray search with errors, see Section 3.2.5, and we have

**Corollary 3.15** *If the robot is able to recognize the start point, in the error model $\ell \in [\,(1-\delta)f, (1+\delta)f\,]$ the doubling strategy $f_i = 2^i$ achieves an optimal competitive factor of $3 + 6\,\frac{1+\delta}{1-\delta}$, and in the error model $\ell \in [\,\frac{1}{1+\delta}f, (1+\delta)f\,]$ a factor of $3 + 6\,(1+\delta)^2$.*

### 3.2.4 Taking the Error into Account

In the previous section, we have seen that a faulty robot is able to reach its goal using the doubling strategy, if its error does not exceed a certain bound. One may guess that the standard doubling strategy is the best we can do in the presence of errors, because it is optimal in the error-free case. Surprisingly, there is a strategy that takes the error $\delta$ into account and yields a competitive factor smaller than the worst-case factor of the doubling strategy we showed in the previous section.

**Theorem 3.16** *In the error model* $\ell \in [\,(1-\delta)f, (1+\delta)f\,]$ *with* $\delta \in [\,0, 1\,[$ *there is a strategy,* $f_i = \left(2\,\frac{1+\delta}{1-\delta}\right)^i$*, that always reaches the goal and achieves a competitive factor of* $1 + 8\left(\frac{1+\delta}{1-\delta}\right)^2$.

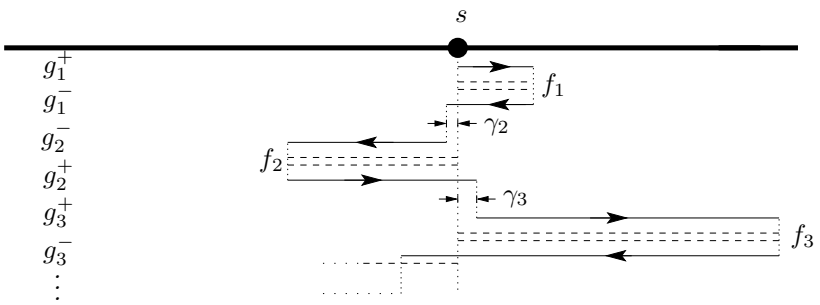



Figure 3.13: An asymmetrical strategy can be turned into a symmetrical strategy (dashed lines).

**Proof.** Let a strategy, $S$, be given by a sequence of nonnegative values, $f_1, f_2, f_3, \ldots$, denoting the nominal values required by the strategy; that is, in the $i$th step the strategy wants the robot to move a distance of $f_i$ to a specified direction—to the right if $i$ is even, and to left if $i$ is odd—and to return to the start point with a movement of $f_i$ to the opposite direction. Remark that every reasonable strategy can be designed this way. Even if we have an asymmetrical strategy $S' = g_1^+, g_1^-, g_2^-, g_2^+, g_3^+, \ldots$ (i. e., a strategy that does not return to start point between two iteration steps), we can turn $S'$ into a symmetrical strategy $S$ by adjusting $f_{i+1}$ with the difference between $g_i^+$ and $g_i^-$. In the example shown in Figure 3.13, let every step towards $s$ be longer than the corresponding step to the opposite site, than $S$ can be described by

$$\begin{array}{lll} f_1 := g_1^+, & & \gamma_2 := g_1^- - g_1^+, \\ f_2 := g_2^- + \gamma_2, & & \gamma_3 := g_2^+ - g_2^- - \gamma_2, \\ f_3 := g_3^+ + \gamma_3 & & \text{etc.} \end{array}$$

As already mentioned, let $\ell_i^+$ and $\ell_i^-$ denote the length of a movement to the right and to the left in the $i$th step, respectively. In the proof of Theorem 3.13 we showed for every online strategy a worst-case ratio of

$$\frac{|\pi_{\text{onl}}|}{d} = 1 + \frac{\sum_{i=1}^{2j+1}(2\ell_i^-)}{\ell_{2j}^+ - \sum_{i=1}^{2j-1}(\ell_i^- - \ell_i^+) + \varepsilon}$$

which achieves its maximum if every step towards the door is as short as possible and every step in the opposite direction is as long as possible; that

is, $\ell_i^- = (1+\delta)\, f_i$ and $\ell_i^+ = (1-\delta)\, f_i$. This yields

$$\frac{|\pi_{\text{onl}}|}{d} = 1 + 2(1+\delta) \cdot \frac{\sum_{i=1}^{2j+1} f_i}{(1-\delta) f_{2j} - 2\delta \sum_{i=1}^{2j-1} f_i + \varepsilon} \,. \tag{3.3}$$

For a fixed $\delta$, 1 and $2(1+\delta)$ are constant, and it is sufficient to find a strategy $S = f_1, f_2, f_3, \ldots$ that minimizes

$$G_{n,\delta}(S) := \frac{\sum_{i=1}^{n+1} f_i}{(1-\delta)\, f_n - 2\delta \sum_{i=1}^{n-1} f_i} \quad \text{for } n > 0 \tag{3.4}$$

and $G_{0,\delta}(S) := \frac{f_1}{1}$; $G_{0,\delta}(S)$ is the worst case after the first iteration step.

Now, we are searching for a strategy $S_\alpha$ in the form $f_i = \alpha^i$ with a fixed $\alpha$—possibly depending on $\delta$—that asymptotically minimizes $G_{n,\delta}(S_\alpha)$. First, we simplify the functional $G$:

$$\begin{aligned}
G_{n,\delta}(S_\alpha) &= \frac{\sum_{i=1}^{n+1} \alpha^i}{(1-\delta)\,\alpha^n - 2\delta \sum_{i=1}^{n-1} \alpha^i} \\
&= \frac{\frac{\alpha^{n+2}-\alpha}{\alpha-1}}{(1-\delta)\,\alpha^n - 2\delta\, \frac{\alpha^n-\alpha}{\alpha-1}} \\
&= \frac{\alpha^2 - \frac{1}{\alpha^{n-1}}}{(\alpha-1)(1-\delta) - 2\delta + \frac{2\delta}{\alpha^{n-1}}} \\
&< \frac{\alpha^2}{(1-\delta)\,\alpha - \delta - 1} =: H_\delta(\alpha)
\end{aligned}$$

To find a minimum of $H_\delta(\alpha)$ we derivate and find the roots

$$\begin{aligned}
H'_\delta(\alpha) &= \frac{2\alpha\,((1-\delta)\,\alpha - \delta - 1) - (1-\delta)\,\alpha^2}{((1-\delta)\,\alpha - \delta - 1)^2} \\
&= \frac{(1-\delta)\,\alpha^2 - 2\,(1+\delta)\,\alpha}{(1-\delta)^2\,\alpha^2 - 2\,(1-\delta^2)\,\alpha + (1+\delta)^2} = 0 \\
&\Leftrightarrow \quad (1-\delta)\,\alpha^2 - 2\,(1+\delta)\,\alpha = 0 \\
&\Leftrightarrow \quad \alpha = 0 \quad \vee \quad \alpha = \frac{2\,(1+\delta)}{1-\delta}
\end{aligned}$$

A strategy with $\alpha = 0$ does not move the robot at all, so $\alpha = 2\,\frac{1+\delta}{1-\delta}$ is the only reasonable root. Note that the denominator of $H'_\delta(2\,\frac{1+\delta}{1-\delta})$ yields $(1+\delta)^2 \neq 0$ for $\delta \geq 0$. To test whether this $\alpha$ is a maximum or minimum, we use the second derivative. We want to evaluate $H''_\delta(\alpha)$ only for the roots

of the numerator of $H'_\delta(\alpha)$; therefore, we can use a simplified form:[6]

$$H''_\delta\Big|_{N'(x)=0}(\alpha) = \frac{2\,(1-\delta)\,\alpha - 2\,(1+\delta)}{(1+\delta)^2}\ .$$

This yields $\frac{2}{1+\delta} > 0$ for $\alpha = 2\,\frac{1+\delta}{1-\delta}$, so we have found a minimum.

To give the competitive factor of our strategy $f_i = \left(2\,\frac{1+\delta}{1-\delta}\right)^i$ we evaluate $H_\delta(\alpha)$. This yields:

$$H_\delta\left(2\,\frac{1+\delta}{1-\delta}\right) = \frac{4\,\left(\frac{1+\delta}{1-\delta}\right)^2}{2\,(1-\delta)\,\frac{1+\delta}{1-\delta} - \delta - 1} = 4\,\frac{1+\delta}{(1-\delta)^2}\ ,$$

and $G_{0,\delta}(S) = 2\,\frac{1+\delta}{1-\delta} \le 4\,\frac{1+\delta}{(1-\delta)^2}$ for $\delta \in [\,0,1\,[$. With Equation 3.3 we get

$$\frac{|\pi_{\mathrm{onl}}|}{d} \le 1 + 2(1+\delta)\cdot 4\,\frac{1+\delta}{(1-\delta)^2} = 1 + 8\,\left(\frac{1+\delta}{1-\delta}\right)^2\ .$$

Note that $1 + 8\,\left(\frac{1+\delta}{1-\delta}\right)^2 \le 1 + 8\,\frac{1+\delta}{1-3\delta}$ holds for $\delta \in [\,0,1\,[$, so our strategy achieves a better competitive factor than the doubling strategy that ignores the presence of errors.

Finally, we check whether our strategy is able to find an arbitrary goal; that is, if the covered distance exceeds the overall drift, see Equation 3.2 on page 90:

$$\begin{aligned}
(1-\delta)\,f_k - \Delta_{k-1} &= (1-\delta)\left(2\,\frac{1+\delta}{1-\delta}\right)^k - 2\delta\,\sum_{i=1}^{k-1}\left(2\,\frac{1+\delta}{1-\delta}\right)^i \qquad (3.5)\\
&= (1-\delta)\left(2\,\frac{1+\delta}{1-\delta}\right)^k - 2\delta\,\frac{\left(2\,\frac{1+\delta}{1-\delta}\right)^k - 2\,\frac{1+\delta}{1-\delta}}{2\,\frac{1+\delta}{1-\delta} - 1}\\
&= \frac{(1+3\delta)\left(2\,\frac{1+\delta}{1-\delta}\right)^k - 2\delta\left(2\,\frac{1+\delta}{1-\delta}\right)^k + 4\delta\,\frac{1+\delta}{1-\delta}}{\frac{1+3\delta}{1-\delta}}\\
&= \frac{\left(\left(2\,\frac{1+\delta}{1-\delta}\right)^k + \delta\left(2\,\frac{1+\delta}{1-\delta}\right)^k + 4\,\frac{1+\delta}{1-\delta}\right)(1-\delta)}{1+3\delta}\\
&> 0 \quad \text{for } \delta \in [\,0,1\,[
\end{aligned}$$

Thus, the robot proceeds towards the door and every goal can be reached.

□

[6] The derivative of a function of type $f(x) = \frac{N(x)}{D(x)}$ is $f'(x) = \frac{D(x)\cdot N'(x) - N(x)\cdot D'(x)}{(D(x))^2}$. If we want to evaluate $f'(x)$ only for the roots of $N(x)$, the derivative simplifies to $f'\big|_{N(x)=0}(x) = \frac{N'(x)}{D(x)}$ [121]. In this case, the denominator of $H''_\delta$ simplifies to $(1+\delta)^2$ for the roots of the numerator of $H'_\delta$.

So far, we have found a strategy that takes the error into account and achieves a better competitive factor than the doubling strategy that is not aware of errors. Now, we want to show that no strategy can achieve a better factor. There are two common ways to show the optimality of a search strategy, which are interesting for themselves, so we discuss both of them.

The first method is to use a theorem by Gal [64] to find a sequence that minimizes a functional that fulfills certain conditions—essentially homogeneity and unimodality.[7] In the proof of Theorem 3.16, we established a functional, Equation 3.3, which describes the competitive factor for every reasonable strategy. To minimize Equation 3.3 for a fixed $\delta$, it is sufficient to minimize the functionals

$$G_{n,\delta}(S) = \frac{\sum_{i=1}^{n+1} f_i}{(1-\delta)\, f_n - 2\delta \sum_{i=1}^{n-1} f_i}\,,$$

see Equation 3.4. To show that these functionals are unimodal, let $C := \max\{G_{n,\delta}(S), G_{n,\delta}(S')\}$ (i. e., $G_{n,\delta}(S) \le C$ and $G_{n,\delta}(S') \le C$). Thus, we have

$$\begin{aligned} \sum_{i=1}^{n+1} f_i &\le C\Big((1-\delta)\, f_n - 2\delta \sum_{i=1}^{n-1} f_i\Big) \quad \text{and} \\ \sum_{i=1}^{n+1} f'_i &\le C\Big((1-\delta)\, f'_n - 2\delta \sum_{i=1}^{n-1} f'_i\Big) \end{aligned}$$

Adding both inequations yields

$$\begin{aligned} & \sum_{i=1}^{n+1} (f_i + f'_i) \le C\Big((1-\delta)\,(f_n + f'_n) - 2\delta \sum_{i=1}^{n-1} (f_i + f'_i)\Big), \\ \iff\ & G_{n,\delta}(S + S') \le C\,. \end{aligned}$$

It can easily be verified that $G_{n,\delta}(S)$ meets also the other requirements. Thus, by Gal's Theorem a function $f_i = \alpha^i$ minimizes $G_{n,\delta}(S)$. Now, we only have to find an appropriate $\alpha$ as we have already done in the proof of Theorem 3.16.

Another way to show the optimality of our strategy is the following. We have seen in the proof of Theorem 3.16 that there is a constant upper bound for $G_{n,\delta}(S)$ that depends on $\delta$. Let $C_\delta(S) := \sup_n G_{n,\delta}(S)$ be the competitive factor of an arbitrary strategy $S$. For a fixed $\delta$, let $S_\delta$ denote a strategy that minimizes $C_\delta(S)$, and let $C^*_\delta$ denote the corresponding factor; that is, $C^*_\delta := C_\delta(S_\delta) = \inf_S C_\delta(S)$. Now, we show that there is always a

[7] That is, the functional fulfills the conditions $F(c \cdot S) = F(S)$ for every constant $c > 0$ and $F(S + S') \le \max\{F(S), F(S')\}$ for every sequences $S, S'$.

strategy $S^*_\delta$ that achieves $C^*_\delta$ not only asymptotically, but exactly *in every step*; that is, $G_{n,\delta}(S^*_\delta) = C^*_\delta$ holds for $n \geq 1$. In the error-free case, for example, the strategy $f_i = (i+1)\,2^i$ achieves the factor 9 exactly in every step. With this, we establish a recurrence and a closed form that describe $S^*_\delta$. Finally, the condition $f_i > 0$ leads to a lower bound for $C^*_\delta$. The idea of using equality is used, for example, in [131] and [122]. Note that both approaches are applicable only for search problems that depend on *one* sequence, $S$. Recently, it was shown that it is possible to combine both approaches to solve a search problem that depends on two independently defined sequences, $S_1$ and $S_2$; for example, a sequence of hit points and a sequence of leave points on rays [105].

**Lemma 3.17** *Let $\ell \in [\,(1-\delta)f, (1+\delta)f\,]$ for $\delta \in [\,0,1\,[$ hold. For every $\delta$ there is a strategy $S^*_\delta$ that achieves $C^*_\delta$ exactly for every $G_{n,\delta}(S^*_\delta)$; that is, $G_{n,\delta}(S^*_\delta) = C^*_\delta$ holds for $n \geq 1$.*

**Proof.** Let $S = (f_1, f_2, f_3, \ldots)$ be a $C^*_\delta$-competitive strategy. We can assume that $S$ is strictly positive; that is, $f_i > 0$ holds. We show that for every $n \geq 1$ there is always a strategy $S^*_\delta$ that fulfills $G_{k,\delta}(S^*_\delta) = C^*_\delta$ for every $1 \leq k \leq n$ by adjusting $S$ adequately.

Deriving the functional

$$G_{n,\delta}(S) = \frac{\sum_{i=1}^{n+1} f_i}{(1-\delta)\,f_n - 2\delta \sum_{i=1}^{n-1} f_i}$$

for $f_n$ yields $\frac{-(1-\delta)\,f_{n+1} - (1+\delta)\,\sum_{i=1}^{n-1} f_i}{\left((1-\delta)\,f_n - 2\delta \sum_{i=1}^{n-1} f_i\right)^2}$; thus, $G_{n,\delta}(S)$ is decreasing in $f_n$.

The denominator of $G_{n,\delta}(S)$ describes the progress towards the door, compare to Equation 3.5. We can assume that $(1-\delta)f_n - 2\delta \sum_{i=1}^{n-1} f_i > 0$ holds; otherwise, the strategy does not increase the search depth in this step and there would be a better strategy.

Now, we can describe $G_{n,\delta}(S)$ by a function $g(f_n) := \frac{f_n + A}{C \cdot f_n - B}$ with $A, B, C > 0$ and $Cf_n - B > 0$. $g(f_n)$ is positive and strictly decreasing for $f_n > 0$. On the other hand, $g(f_n)$ goes to infinity, if $f_n$ decreases towards $f_n = \frac{B}{C}$. Altogether, we can increase $G_{n,\delta}(S)$ continuously to any desired value by decreasing $f_n$ adequately.

Further, it is easy to see that $G_{k,\delta}(S)$ is increasing in $f_n$ for every $k \neq n$:

- For $k < n-1$, $G_{k,\delta}(S)$ is not affected by $f_n$ at all.
- For $k = n-1$, $f_n$ appears only in the numerator of $G_{k,\delta}(S)$, which increases if $f_n$ grows.
- For $k > n$ the numerator increases and the denominator shrinks for a growing $f_n$ because of the coefficient $-2\delta$.

Vice versa, $G_{k,\delta}(S)$ is decreasing in $f_n$ for every $k \neq n$ if $f_n$ reduces. Thus, by shrinking $f_n$ we decrease every $G_{k,\delta}(S)$ for $k \neq n$.

Now, we show by an induction over $n$ that we can decrease our strictly positive, $C^*_\delta$-competitive strategy $S$ to a strictly positive strategy $S^*_\delta$, which fulfills $G_{k,\delta}(S^*_\delta) = C^*_\delta$ for every $1 \leq k \leq n$. Additionally, $S^*_\delta$ is equal to $S$ for every $f_i$ with $i \geq n+1$.

For $n = 1$ let us assume that $G_{1,\delta}(S) \leq C^*_\delta$ holds; otherwise we are done. Using the preceding argumentation, we decrease $f_1$ by a small value $\varepsilon > 0$ to $f'_1 := f_1 - \varepsilon > 0$ such that $G_{1,\delta}(S') = C^*_\delta$ holds. Every $G_{k,\delta}(S')$ with $k \neq 1$ decreases; thus, the new strategy is still $C^*_\delta$-competitive. $f'_2$ is strictly positive, so $f'_1$ has to be strictly positive; otherwise, the numerator of $G_{1,\delta}(S')$ decreases, and, in turn, $G_{1,\delta}(S')$ decreases.

For the induction step, we assume that $G_{k,\delta}(S) = C^*_\delta$ holds for every $1 \leq k \leq n$. Let $G_{n+1,\delta}(S) < C^*_\delta$; otherwise we are done. As in the previous case, we adjust $S$ to $S^{(1)}$ by decreasing $f_{n+1}$ to $f^{(1)}_{n+1} := f_{n+1} - \varepsilon$ such that $G_{n+1,\delta}(S^{(1)}) = C^*_\delta$ holds. With the preceding argumentation we know that $G_{k,\delta}(S^{(1)})$ decreases for $k \neq n$. Again, $f^{(1)}_{n+1}$ has to be strictly positive, because $f^{(1)}_{n+2}$ is strictly positive.

Now, unfortunately, $G_{n,\delta}(S^{(1)}) < C^*_\delta$ holds and we have to apply the induction hypothesis again: We adjust $S^{(1)}$ to $S^{(2)}$ by decreasing $f^{(1)}_n$ again such that $G_{k,\delta}(S^{(2)}) = C^*_\delta$ holds for every $1 \leq k \leq n$. Again, $S^{(2)}$ is strictly positive. Now, in turn, we have $G_{n+1,\delta}(S^{(2)}) < C^*_\delta$ and we have to adjust $S^{(2)}$ again by decreasing $f^{(2)}_{n+1}$ adequately.

Altogether, the process described previously generates for every element $f_i$, $1 \leq i \leq n+1$, of $S$ a sequence of values $f^{(k)}_i$. This sequence is strictly decreasing, whereas every $f^{(k)}_i$ is strictly positive; thus, the sequence converges towards an unique, positive limit. Let $S^*_\delta$ be the sequence of limit values; that is, $f^*_i := \lim_{k\to\infty} f^{(k)}_i$. $S^*_\delta$ fulfills $G_{k,\delta}(S^*_\delta) = C^*_\delta$ for $1 \leq k \leq n+1$, which finishes the induction and the proof. □

Now, we can assume that there is an optimal strategy $S^*_\delta$ with $G_{n,\delta}(S^*_\delta) = C^*_\delta$ for every $n \geq 1$, and give a lower bound for the competitive factor.

**Theorem 3.18** *In the error model $\ell \in [\,(1-\delta)f, (1+\delta)f\,]$ for $\delta \in [\,0,1\,[$, no strategy for searching a point on a line yields a competitive factor smaller than*

$$1 + 8\left(\frac{1+\delta}{1-\delta}\right)^2 .$$

**Proof.** We describe an optimal strategy $S^*_\delta$ that fulfills $G_{n,\delta}(S^*_\delta) = C^*_\delta$ for every $n \geq 1$ by a recurrence. From $G_{n-1,\delta}(S^*_\delta) = G_{n-2,\delta}(S^*_\delta) = C^*_\delta$ we

conclude

$$\sum_{i=1}^{n} f_i^* = C_\delta^* \left( (1-\delta) f_{n-1}^* - 2\delta \sum_{i=1}^{n-2} f_i^* \right) \text{ and}$$

$$\sum_{i=1}^{n-1} f_i^* = C_\delta^* \left( (1-\delta) f_{n-2}^* - 2\delta \sum_{i=1}^{n-3} f_i^* \right) .$$

Subtracting both yields for all $n \geq 3$

$$f_n^* = C_\delta^* (1-\delta) f_{n-1}^* - C_\delta^* (1+\delta)\, f_{n-2}^* \tag{3.6}$$

This equation is a linear, homogeneous recurrence of degree $r = 2$ and can be solved using standard techniques as described, for example, in Graham et al. [68]. First, we find the *characteristic polynom*, $\chi(t)$, by building a generating function, $\mathcal{F}(t)$, and "reflecting" the denominator of $\mathcal{F}(t) = \frac{N(t)}{D(t)}$; that is, $\chi(t) = t^r \cdot D(\frac{1}{t})$. This yields

$$\chi(t) = t^2 - C_\delta^* \,(1-\delta)\, t + C_\delta^* \,(1+\delta) \,.$$

$\chi(t)$ has the roots

$$z, \overline{z} = \frac{1}{2} \left( C_\delta^* \,(1-\delta) \pm \sqrt{C_\delta^* \left( C_\delta^* \,(1-\delta)^2 - 4\,(1+\delta) \right)} \right) .$$

where $\overline{z}$ denotes the conjugate of $z$. With this, the recurrence is given in the closed form:[8]

$$f_n^* = \lambda\, z^n + \overline{\lambda}\, \overline{z}^n = 2\,\mathrm{Re}(\lambda\, z^n).$$

$\lambda$ and $\overline{\lambda}$ are determined by the equations

$$\begin{aligned} f_1^* &= \lambda + \overline{\lambda} \quad \text{and} \\ f_2^* &= \lambda z + \overline{\lambda}\overline{z}\,, \end{aligned}$$

and therefore by the starting values $f_1^*$ and $f_2^*$.

Now, we use a property of complex numbers to establish a lower bound for $C_\delta^*$, see [82, 115]. We can represent complex numbers $z = a + b\mathrm{i}$ by polar coordinates $z = |z| \cdot \mathrm{e}^{\mathrm{i}\varphi}$ with $\varphi = \arg(z)$, and multiply two numbers $z_1$ and $z_2$ by $z_1 \cdot z_2 = |z_1| \cdot |z_2| \cdot \mathrm{e}^{\mathrm{i}(\varphi_1 + \varphi_2)}$; thus, essentially, we add the angles the complex numbers form with the positive $X$ axis.

If the radiant, $C_\delta^* \left( C_\delta^* (1-\delta)^2 - 4(1+\delta) \right)$, of $z$ is negative, then $z$ is not real and $\arg(z) \neq 0$ holds. In this case, there exists a smallest natural number $k$ such that $\arg(z^k) \geq \frac{\pi}{2}$ holds; that is, $\lambda z^k$ lies in the left halfplane $\{X \leq 0\}$. Thus, also the conjugate of $\lambda z^k$ lies in the left halfplane and $f_k = \lambda z^k + \overline{\lambda}\overline{z}^k$ becomes zero or negative, which is a contradiction because the $f_i^*$'s have to be positive for any optimal strategy $S^*$. The roots of the radiant

[8] $\mathrm{Re}(z)$ denotes the real part, $b$, of a complex number $z = a + b\mathrm{i}$.

are 0 and $4\,\frac{1+\delta}{(1-\delta)^2}$. Thus, $f_k^*$ becomes negative or zero, if $C_\delta^* < 4\,\frac{1+\delta}{(1-\delta)^2}$ holds. With Equation 3.3 on page 93 this yields an overall competitive factor of at least

$$1 + 8\left(\frac{1+\delta}{1-\delta}\right)^2,$$

which exactly matches the factor of the strategy described in Theorem 3.16. □

Remark that the proof also holds for $\delta = 0$, which gives another proof of the factor 9 for the optimal strategy in the error-free case. Thus, line search with errors is generalized adequately.

Now, we want to state similar results in the second error model. We use the same techniques as in the previous proofs, so we give only a brief overview.

**Corollary 3.19** *In the error model $\ell \in [\,\frac{1}{(1+\delta)}f, (1+\delta)f\,]$ for $\delta > 0$ there is a strategy, $f_i = \left(2(1+\delta)^2\right)^i$, that always reaches the goal and achieves an optimal competitive factor of*

$$1 + 8(1+\delta)^4\,.$$

**Proof.** We proceed exactly as in the proofs of Theorem 3.16, Lemma 3.17 and Theorem 3.18. Again, the competitive factor achieves its maximum if every step towards the door is as short as possible and every step in the opposite direction is as long as possible; that is, we set $\ell_i^- = (1+\delta)\,f_i$ and $\ell_i^+ = \frac{1}{1+\delta}\,f_i$ in this error model. Analogously to Equation 3.3 on page 93 this gives a competitive factor of

$$\begin{aligned}
\frac{|\pi_{\text{onl}}|}{d} &= 1 + \frac{\sum_{i=1}^{2j+1}(2\ell_i^-)}{\ell_{2j}^+ - \sum_{i=1}^{2j-1}(\ell_i^- - \ell_i^+) + \varepsilon} \\
&= 1 + \frac{\sum_{i=1}^{2j+1}(1+\delta)\,f_i}{\frac{f_{2j}}{1+\delta} - \sum_{i=1}^{2j-1}\left(1+\delta-\frac{1}{1+\delta}\right)f_i + \varepsilon} \\
&= 1 + 2\,(1+\delta)^2 \cdot \frac{\sum_{i=1}^{2j+1} f_i}{f_{2j} - \delta\,(2+\delta)\,\sum_{i=1}^{2j-1} f_i + \varepsilon}
\end{aligned}$$

1 and $2\,(1+\delta)^2$ are constant for a fixed $\delta$, so it suffices to consider the functionals—compare Equation 3.4—

$$G_{n,\delta}(S) := \frac{\sum_{i=1}^{n+1} f_i}{f_n - \delta(2+\delta)\sum_{i=1}^{n-1} f_i} \qquad \text{for } n > 1\,.$$

We search for a strategy $f_i = \alpha^i$ that minimizes

$$\begin{aligned}
G_{n,\delta}(S_\alpha) &= \frac{\sum_{i=1}^{n+1}\alpha^i}{\alpha^n - \delta(2+\delta)\sum_{i=1}^{n-1}\alpha^i} = \frac{\alpha^2 - \frac{1}{\alpha^{n-1}}}{\alpha - 1 - 2\delta - \delta^2 + \frac{\delta\,(2+\delta)}{\alpha^{n-1}}} \\
&< \frac{\alpha^2}{\alpha - 1 - 2\delta - \delta^2} \;=:\; H_\delta(\alpha)
\end{aligned}$$

by deriving $H_\delta(\alpha)$ and finding the roots:

$$H'_\delta(\alpha) = \frac{\alpha\,(\alpha - 2 - 4\delta - 2\delta^2)}{(\alpha - 1 - 2\delta - \delta^2)^2} = 0$$
$$\Leftrightarrow \quad \alpha = 0 \quad \vee \quad \alpha = 2\delta^2 + 4\delta + 2 = 2(1+\delta)^2$$

This is a minimum, because

$$H''_\delta\big|_{N'(x)=0}(\alpha) = \frac{2\alpha - 2 - 4\delta - 2\delta^2}{(\alpha - 1 - 2\delta - \delta^2)^2} \qquad \text{and}$$
$$H''_\delta\big|_{N'(x)=0}\left(2\delta^2 + 4\delta + 2\right) = \frac{2\delta^2 + 4\delta + 2}{(\delta^2 + 2\delta + 1)^2} > 0$$

$H_\delta\left(2\delta^2 + 4\delta + 2\right)$ yields $4\,(1+\delta)^2$, so we get an overall competitive factor of

$$\frac{|\pi_{\text{onl}}|}{d} \le 1 + 8\,(1+\delta)^4\,.$$

The strategy proceeds in every iteration step at least by

$$\begin{aligned}
\frac{1}{1+\delta} f_n - \Delta_n &= \frac{\alpha^n}{1+\delta} - \frac{\delta\,(2+\delta)}{1+\delta} \cdot \sum_{i=1}^{n-1} \alpha^i \\
&= \frac{\alpha^n}{1+\delta} - \frac{\delta\,(2+\delta)}{1+\delta} \cdot \frac{\alpha^n - \alpha}{\alpha - 1} \\
&= \frac{(1 + 4\delta + 2\delta^2)\,\alpha^n - \delta\,(2+\delta)\,\alpha^n + \delta\,(2+\delta)\,\alpha}{(1+\delta)(1+4\delta+2\delta^2)} \\
&= \frac{(1 + 2\delta + \delta^2)\,\alpha^n + \delta\,(2+\delta)\,\alpha}{(1+\delta)(1+4\delta+2\delta^2)} > 0 \quad \text{for } \delta > 0
\end{aligned}$$

and the strategy reaches every goal.

Finally, we show that the given strategy is optimal. Lemma 3.17 holds also for this model. With the same technique we adjust a given strategy such that the optimal factor is achieved exactly in every step. From $G_{n-1,\delta}(S^*_\delta) = G_{n-2,\delta}(S^*_\delta) = C^*_\delta$ we conclude

$$\sum_{i=1}^{n} f^*_i = C^*_\delta \left( f^*_{n-1} - \delta(2+\delta) \sum_{i=1}^{n-2} f^*_i \right) \text{ and}$$
$$\sum_{i=1}^{n-1} f^*_i = C^*_\delta \left( f^*_{n-2} - \delta(2+\delta) \sum_{i=1}^{n-3} f^*_i \right).$$

Thus, analogously to Equation 3.6, an optimal strategy is described by the recurrence

$$f^*_n = C^*_\delta\, f^*_{n-1} - C^*_\delta\,(1+\delta)^2\, f^*_{n-2} \quad \text{for } n \ge 3\,,$$

and the characteristic polynom $\chi(t) = t^2 - C_\delta^* t + C_\delta^*(1+\delta)^2$ has the roots

$$z, \bar{z} = \frac{1}{2}\left(C_\delta^* \pm \sqrt{C_\delta^*\,(C_\delta^* - 4(1+\delta)^2)}\right).$$

The radiant is nonnegative for $C_\delta^* \geq 4(1+\delta)^2$. With the same arguments as in the proof of Theorem 3.18 we conclude that the competitive factor is at least $1 + 8(1+\delta)^4$. □

### 3.2.5 Error-Prone Searching on $m$ Rays

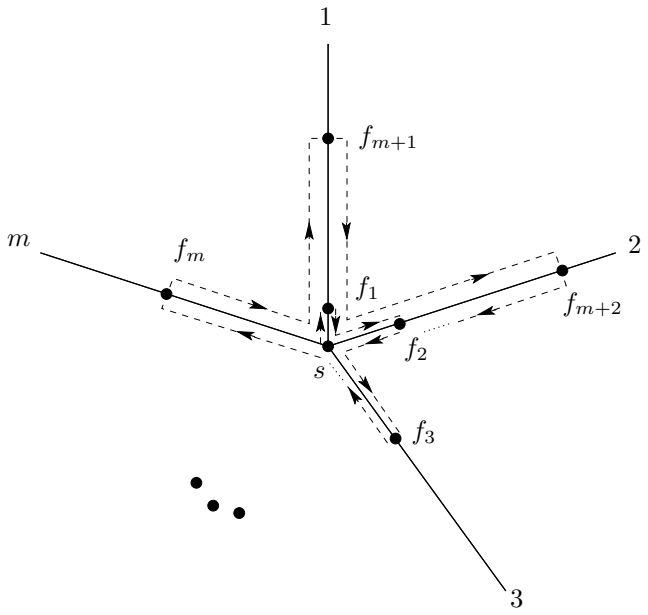



Figure 3.14: Searching on $m$ rays.

In the previous sections we considered the search for a goal on an infinite line. This can be generalized to a search on $m$ infinite rays emanating from the robot's start point. The target is located on one of the rays, but—as in the previous case—the searcher neither knows the ray containing the target nor the distance to the target.

It was shown by Gal [64] that w. l. o. g. the rays can be visited in a cyclic order and with increasing depth. Strategies with this properties are called *periodic* and *monotone*. We can describe such strategies by a sequence of values $f_1, f_2, \ldots$ with $f_i < f_{i+m}$. In the $i$th step, the searcher visits the ray $i \bmod m$ up to the distance $f_i$ and returns to the start point, see Figure 3.14. An optimal strategy is described by $f_i = \left(\frac{m}{m-1}\right)^i$ and achieves a competitive factor of $1 + 2m\left(\frac{m}{m-1}\right)^{m-1}$ [64, 11]

Even if the robot's movement is erroneous, the robot is able to return to its start point, $s$, because this is the only point in the environment where the rays meet. The robot has to recognize this point; otherwise, we cannot guarantee that all rays are visited. However, the search depth may vary from the nominal value. These variations may influence the correctness and the competitive factor of a strategy. Let us first assume that the error, $\delta$, is known to the strategy. Surprisingly, it turns out that we do not have to distinguish whether $\delta$ is known or unknown to the strategy.

**Theorem 3.20** *Let the distance covered by the robot, $\ell$, be in the range $[\,(1-\delta)f, (1+\delta)f\,]$ for $\delta \in [\,0, 1\,[$. Searching for a target located on one of $m$ rays using a monotone and periodic strategy is competitive with an optimal factor of*

$$3 + 2\,\frac{1+\delta}{1-\delta}\,\left(\frac{m^m}{(m-1)^{m-1}} - 1\right)$$

*for $\delta < \frac{e-1}{e+1}$.*

**Proof.** Let $S$ be a periodic and monotone strategy given by a sequence of values $f_1, f_2, f_3, \ldots$, and let $\ell_i$ denote the distance covered by the robot in the $i$th step. Similar to the search on a line, we establish a functional that describes the worst case of any possible strategy and look for a sequence that minimizes this functional. As in the previous sections, we achieve the worst case if the target is slightly missed in step $k$, but hit in step $k+m$. This yields a worst-case ratio of

$$\frac{|\pi_{\text{onl}}|}{d} = 1 + \frac{2\sum_{i=1}^{k+m-1}\ell_i}{\ell_k + \varepsilon}. \tag{3.7}$$

This ratio achieves its maximum, if we maximize every $\ell_i$ for $i \neq k$ and choose a worst-case value for $\ell_k$. Therefore, we set $\ell_k := (1+\beta)\,f_k$ with $\beta \in [-\delta, \delta]$, and $\ell_i := (1+\delta)\,f_i$ for $i \neq k$. This yields[9]

$$\begin{aligned}
\frac{|\pi_{\text{onl}}|}{d} &= 1 + \frac{2}{(1+\beta)\,f_k}\left((1+\beta)\,f_k + (1+\delta)\sum_{\substack{i=1,\ldots,k+m-1\\ i\neq k}} f_i\right)\\
&= 1 + \frac{2}{(1+\beta)\,f_k}\left((1+\beta)\,f_k + (1+\delta)(\sum_{i=1}^{k+m-1} f_i) - (1+\delta)\,f_k\right)\\
&= 1 + 2 + \frac{2(1+\delta)\,\sum_{i=1}^{k+m-1} f_i}{(1+\beta)\,f_k} - 2\,\frac{1+\delta}{1+\beta}\\
&= 3 + 2\,\frac{1+\delta}{1+\beta}\,\left(\frac{\sum_{i=1}^{k+m-1} f_i}{f_k} - 1\right).
\end{aligned}$$

[9]For convenience we omit $\varepsilon$ in the following.

Obviously, this ratio achieves its maximum for $\beta = -\delta$. Thus, we want to find a sequence that minimizes $3+2\,\frac{1+\delta}{1-\delta}\left(\frac{\sum_{i=1}^{k+m-1} f_i}{f_k} - 1\right)$, which, in turn, amounts to consider the functionals

$$G_k(S) := \frac{\sum_{i=1}^{k+m-1} f_i}{f_k},$$

because $2\,\frac{1+\delta}{1-\delta}$ is constant for a fixed $\delta$. These functionals are identical to the functionals considered in the error-free $m$-ray search; thus, we can use the results known for this case. That is, the strategy $f_i = \left(\frac{m}{m-1}\right)^i$ yields an optimal upper bound of $G_k(S) < \frac{m^m}{(m-1)^{m-1}}$, see [11, 64]. Altogether, we get the worst-case ratio

$$3 + 2\,\frac{1+\delta}{1-\delta}\left(\frac{m^m}{(m-1)^{m-1}} - 1\right),$$

and the factor as well as the optimality are shown. Remark that the optimal strategy does not depend on $\delta$; thus, we do not have to distinguish whether the error range is known or unknown to the strategy.

Finally, we have to ensure that our strategy is still monotone even if the robot's movement is erroneous. Thus, we have to make sure that the maximal covered distance in the step $f_{k-m}$ does not exceed the minimal covered distance in the step $f_k$; that is, $(1-\delta)\,f_k > (1+\delta)\,f_{k-m}$ holds. For the strategy $f_i = \left(\frac{m}{m-1}\right)^i$ we get

$$\begin{aligned}(1-\delta)\left(\frac{m}{m-1}\right)^k &> (1+\delta)\left(\frac{m}{m-1}\right)^{k-m} \\ \Leftrightarrow \delta &< \frac{\left(\frac{m}{m-1}\right)^m - 1}{\left(\frac{m}{m-1}\right)^m + 1} =: \delta_{\max}(m).\end{aligned}$$

$\delta_{\max}(m)$ converges to $\frac{e-1}{e+1} \approx 0.4621$ (from above) for $m \to \infty$. Thus, we can be sure that our strategy is monotone, if $\delta < \frac{e-1}{e+1}$ holds. □

**Corollary 3.21** *Let the distance covered by the robot, $\ell$, be in the range $[\,\frac{1}{1+\delta}f, (1+\delta)f\,]$ for $\delta > 0$. Searching for a target located on one of $m$ rays using a monotone and periodic strategy is competitive with an optimal factor of*

$$3 + 2\,(1+\delta)^2\left(\frac{m^m}{(m-1)^{m-1}} - 1\right)$$

*for $\delta < \sqrt{e} - 1$.*

**Proof.** With the same arguments as in the proof of Theorem 3.20 the worst-case ratio of Equation 3.7 is maximized for $\ell_k := \frac{1}{1+\beta}\, f_k$ with $\beta \in [0, \delta]$, and $\ell_i := (1+\delta)\, f_i$ for $i \neq k$. This yields

$$\begin{aligned} \frac{|\pi_{\text{onl}}|}{d} &= 1 + 2\,(1+\beta)\,\frac{(1+\delta)\Big(\sum_{i=1}^{k+m-1} f_i\Big) - (1+\delta)\, f_k + \frac{1}{1+\beta}\, f_k}{f_k} \\ &= 3 + 2\,(1+\beta)(1+\delta)\left(\frac{\sum_{i=1}^{k+m-1} f_i}{f_k} - 1\right). \end{aligned}$$

This maximizes for $\beta = \delta$, and we get the same functionals and the same strategy as in the previous case.
To ensure monotonicity we have to fulfill $\frac{1}{1+\delta}\left(\frac{m}{m-1}\right)^k > (1+\delta)\left(\frac{m}{m-1}\right)^{k-m}$, which is equivalent to $\delta < \sqrt{\left(\frac{m}{m-1}\right)^m} - 1 \leq \sqrt{e} - 1$. □

## 3.3 Summary

We have seen that is possible to incorporate erroneous behaviors into the robot model while we are still able to give theoretically funded analysis. We considered the Pledge algorithm under errors in sensors and motion and established sufficient requirements for the robot. Particularly, we showed that a robot equipped with a simple compass will fulfill this task. For an implementation of an error-prone Pledge algorithm see [78].

Further, we analyzed the usual doubling strategy, $f_i = 2^i$, for reaching a door along a wall in the presence of errors in movements. In the error model $[(1-\delta)f_i, (1+\delta)f_i\,], \delta \in [0, 1[$, we showed that the robot is still able to reach the door if the error $\delta$ not greater than $\frac{1}{3}$. Moreover, if the error not greater than $\frac{1}{4}$ the robot will need at most two iteration steps more than the error-free robot. Both error bounds are rather big—33 percent and 25 percent—, so it can be expected that real robots will meet this error bounds. The competitive factor of this strategy is given by $8\frac{1+\delta}{1-3\delta} + 1$.

If the maximal error is known to the strategy, there is a strategy that takes the error into account and achieves a better factor than the strategy $f_i = 2^i$. The strategy $f_i = \left(2\,\frac{1+\delta}{1-\delta}\right)^i$ achieves the optimal competitive factor of $1 + 8\left(\frac{1+\delta}{1-\delta}\right)^2$. Table 3.1 shows the competitive factors for several given error bounds.

The case of $m$ rays the problem is easier to solve, because the robot detects the start point between two iterations. If the error $\delta$ is not greater than $\delta_{\max}(m) = \frac{\left(\frac{m}{m-1}\right)^m - 1}{\left(\frac{m}{m-1}\right)^m + 1}$ —which is less than $\frac{e-1}{e+1} \approx 0.46$ for every $m$—the standard $m$-ray doubling strategy with $f_i = \left(\frac{m}{m-1}\right)^i$ is the optimal periodic and monotone strategy and yields a factor $3 + 2\frac{1+\delta}{1-\delta}\left(\frac{m^m}{(m-1)^{m-1}} - 1\right)$. In this

setting it may be interesting to consider turning errors that may prevent the robot from entering the next corridor correctly.

| | competitive factor | | |
|---|---|---|---|
| | line search | | $m$-ray search |
| $\delta$ | $\delta$ unknown | $\delta$ known | |
| 0 | 9 | 9 | $2em+1$ |
| 0.1 | $\approx 13.571$ | $\approx 12.95$ | $\approx 2.44em+0.55$ |
| 0.2 | 25 | 19 | $3em$ |
| 0.25 | 41 | $\approx 23.22$ | |
| 0.3 | 105 | $\approx 28.59$ | |
| 0.33 | 1065 | $\approx 32.52$ | |
| 1/3 | — | 33 | $4em-1$ |
| 0.46 | — | | $\approx 5.41em-2.41$ |
| 1/2 | — | 73 | |
| 0.9 | — | 2889 | |

Table 3.1: Competitive factor for several error bounds in the error model $[(1-\delta)f_i, (1+\delta)f_i\,], \delta \in [0,1\,[$.

However, we considered only two very easy tasks in robot motion planning. Now, of course, it is interesting to know how other theoretically well-understood algorithms behave in the presence of errors. The exploration of simple polygons using the strategy *PolyExplore* by Icking et al. [84] may be very useful to the robotics community, so it might be worthwhile to consider the influence of errors to this algorithm. Another concern is the behavior of algorithms designed for orthogonal scenes in $\delta$-pseudo-orthogonal scenes. Especially, it is interesting to know how the exploration strategy by Deng et al. [43] adapts to $\delta$-pseudo-orthogonal scenes. As for the searching on a line, we can consider two different tasks in this case. First, we may consider a strategy that is not aware of deviations from the orthogonality, and ask for the performance of the exploration strategy by Deng et al. depending on $\delta$. The second question is, how an exploration strategy benefits from knowing the maximal divergence, $\delta$. Or, similarly, we may ask whether PolyExplore benefits from the knowledge that angles in its surroundings cannot be arbitrarily small.

Concerning mazes it might be interesting to know how to escape from an unknown maze with one-way roads. Imagine, you are leaving the old quarter of a large city. The city is surrounded by a ring of larger streets. As soon as you reach this ring, there are traffic signs that lead you to your destination, but you have no clue how to get there. To make things worse, there are many one-way roads in the old quarter. We can model this problem, for example, by adding directed edges between obstacle boundaries. The searcher

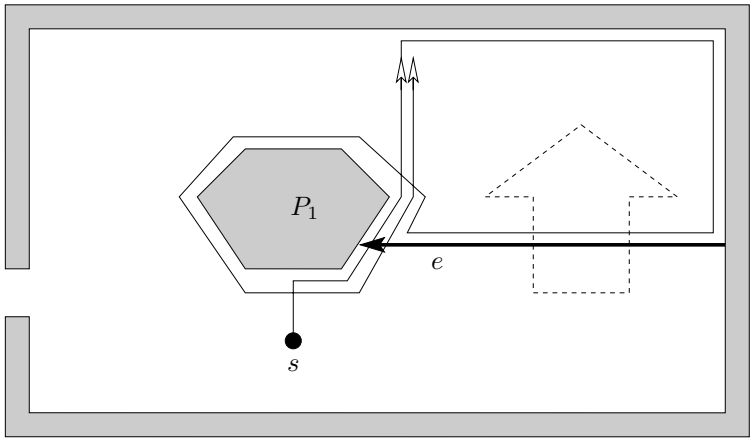



Figure 3.15: Applying the Pledge algorithm to environments with one-way roads does not work.

is allowed to cross this edges only from the left to the right—seen in the direction of the edge—but never the opposite way. It is easy to see that we cannot apply the Pledge algorithm treating the ends of one-way roads as obstacle edges if we encounter them from the wrong side. Figure 3.15 shows an example with one one-way road $e$. A robot starting in $s$ hits the obstacle $P_1$, passes $e$, and leaves $P_1$. Following the second obstacle, the robot meets the exit side of $e$. Thus, it follows $e$ and circles $P_1$ again. The angle counter gets zero in the same vertex of $P_1$ as in the first visit and the robot is trapped in an endless loop. Therefore, the simple treatment of one-way roads does not work and we a more sophisticated strategy.

# Chapter 4

# Optimal Search Paths

We have seen that exploration and searching are fundamental tasks in online motion planning and well-studied in many settings. Both tasks seem to be closely related: Every search strategy has to ensure that every point in the environment is inspected; thus, every search strategy is also an exploration strategy. However, there is a fundamental difference between searching and exploration: An online exploration strategy competes with the shortest offline exploration path, whereas an online search strategy is compared to the shortest path from the start to the goal. Therefore, it is necessary for a search strategy to explore the environment in a breadth-first manner to achieve a (good) competitive factor; that is, the strategy has to scan the environment located closely to start point before it deals with locations further away. In the previous chapter, for example, we have seen search strategies for a point on a line and on $m$ rays. Both strategies iteratively increase the search depth.

On the other hand, it is more reasonable for an exploration strategy to explore the environment in a depth-first manner to avoid unnecessary routes; thus, an exploration completely explores areas farther away before it returns to the start. For example, the strategies SmartDFS and CellExplore we have seen in Chapter 2 proceed that way.

Moreover, there are environments that can be explored with a constant competitive factor although no search strategy is able to achieve such a factor; see, for example, the simple polygon shown in Figure 4.1: The polygon with $n$ vertices consists of $\frac{n}{4}$ corridors of length 1 ending with a little pocket. One of the pockets contains the target. In the worst case, the target is located in the last visited corridor, yielding a path length in $\Omega(n)$, whereas the length of the shortest path from $s$ to $t$ is $1 + \varepsilon$.

Although in such a setting no search strategy can compete with the optimal solution in terms of the competitive framework, there still may be paths through the environment that are better suited for searching than others. The competitive ratio is no appropriate measure of quality; therefore,

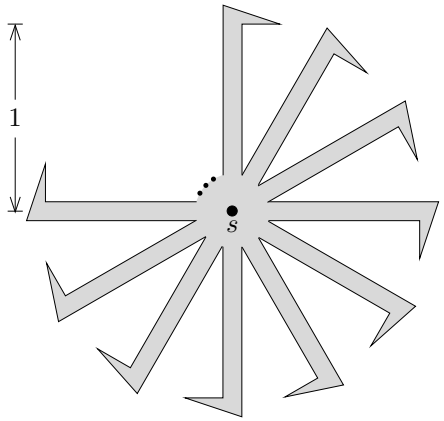


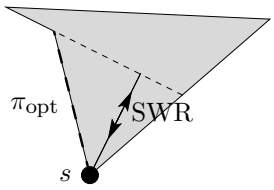

Figure 4.1: No search strategy achieves a competitive factor better than $\Omega(n)$ [115].

we need to find another way to rate search paths.

In this chapter, we consider the *search ratio* as a measure of quality. The search ratio was introduced by Koutsoupias et al. [118] and considered by Fleischer et al. [59] for different settings. The quality of a search path is determined by a worst-case target point—a point that maximizes among all target points, $t$, the ratio between the length of the searcher's path up to $t$ and the shortest path to $t$, see Definition 4.1. An *optimal search path* has the minimal search ratio among all search paths in the given environment. The *optimal search ratio*—the search ratio of the optimal search path—is an appropriate measure for the *searchability* of an environment. Thus, a "good" search path is a path that achieves a good approximation of the optimal search path.

For some types of environments it is easy to find optimal search paths even online, because every optimal competitive search strategy computes an optimal search path. Such strategies are known, for example, for searching a point on a line or on $m$ rays [64, 11], see also Section 3.2; for searching in a special class of simple polygons called *streets*, see Klein [113] and Icking et al. [94]; or a special case of searching for a ray, the *window shopper problem*,[1] see Eubeler et al. [54]. However, for other types of environments it is hard to find an optimal search path even for easy examples, and even in the offline case, see [118, 59]. For instance, in a simple polygon we obviously have to visit the *essential cuts*, see Section 4.2.2. But even if we have given the optimal visiting order for the cuts, it is not clear, how to visit them.

[1] A buyer walks along a line of shopping windows, looking for a present. As soon as an appropriate item is sighted, the buyer walks towards the window. Thus, the task is to find the origin (the item) of a ray, $r$ (the line of sight), that is located on another ray, $r'$ (the shopping windows). $r'$ is perpendicular to $r$, its position is known, and the searcher's start point and $r$ are located on the same side of $r'$.




Intuitively, one may guess that the cuts are visited with the law of reflection[2] similar to the shortest watchman route, see Chin and Ntafos [32], but Eubeler [53] found a counterexample to this conjecture: In the figure, the optimal search path, $\pi_{\text{opt}}$, moves directly to the corner of the polygon.

Therefore, we are interested in (online) approximations of the optimal search path. Note that both in the online and in the offline setting the location of the target point is unknown. In the offline setting, the environment is known in advance to the agent; this information is not provided to an online searcher.

It turns out that there is a close relation between exploration—more precisely *depth-restricted* exploration—and searching; that is, there is a strategy for approximating the optimal search path up to a constant factor for a given setting, if there is a constant-competitive, depth-restricted exploration algorithm in this setting, see Section 4.2. For online settings fulfilling a certain condition, it can even be shown that there is an approximation if *and only if* there is such an exploration algorithm, see Section 4.3. We call environments where no constant-factor online approximation of the optimal search path exists—for short—*hard-searchable environments*.

Although our interest is mainly in polygons, we present an approximation framework and a generalized lower bound that can also be applied to other types of environments such as trees or graphs.

## 4.1 Definitions

We want to find a good search path in some given environment, $\mathcal{E}$, of arbitrary type: a polygon, a tree, a graph, or a more special subclass of these types such as $m$ rays emanating from a common point. The only restriction to the type of environments is that we require that there is a shortest path from every point, $p$, in $\mathcal{E}$ back to the start point, $s$, that is of the same length as a shortest path from $s$ to $p$;[3] that is,

$$\forall p \in \mathcal{E} : |\operatorname{sp}(s,p)| = |\operatorname{sp}(p,s)| .$$

In most cases, we want to search the whole environment, but there are special kinds of search problems where we know that the goal may be hidden only in some parts of the scene. Therefore, let the *goal set* $\mathcal{G} \subseteq \mathcal{E}$ denote the part of the environment where the goal may be located. For example, if we search in a graph, $G = (V, E)$, the goal may be located anywhere in the

[2] That is, the *incoming angle* is equal to the *outgoing angle*, see Figure 4.3.

[3] This condition excludes directed graphs, but this is no severe restriction, because there is no search path approximation with a constant factor for directed graphs, see [59].

graph; we call this setting *geometric search* and set $\mathcal{G} := V \cup E$. In contrary, we may consider a *vertex search*, where the goal is restricted to be hidden in a vertex; in this case, we set $\mathcal{G} := V$. Thus, we have to inspect all potential goal locations in $\mathcal{G}$. We require—as usual—that the distance to the goal is at least 1. Otherwise, no search strategy is able to achieve a competitive factor.[4]

The searcher can either be *blind* (i. e., it can sense only its very close neighborhood) or it can have *vision*; that is, it can see objects far away from its current position, if the line of sight is not blocked by an obstacle. We assume that the searcher is equipped with enough memory to store a map of its environment, and its movements, sensors, map making, and localization is error-free.

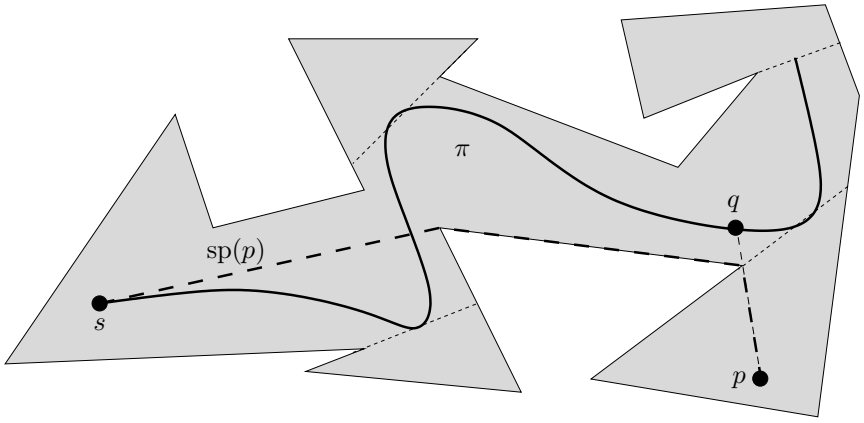



Figure 4.2: A search path, $\pi$, inside a polygon. Moving along $\pi$, the searcher sees $p$ for the first time from $q = p_\pi$. The dashed path shows the shortest path from $s$ to the goal $p$.

We use the following notations: Given a start point, $s \in \mathcal{E}$, let $\pi$ be a path in the environment $\mathcal{E}$ starting in $s$. For a given point $q \in \pi$ let $\pi(q)$ denote the part of $\pi$ from $s$ to $q$. For an arbitrary point $p \in \mathcal{E}$ let $\mathrm{sp}(p)$ denote a shortest path from $s$ to $p$ in the given environment, and $p_\pi$ denote the point $q \in \pi$ from which a searcher following $\pi$ sees $p$ for the first time, see Figure 4.2. The quality of a search path is measured by its *search ratio*:

**Definition 4.1** Let $\mathcal{E}$ be an environment, $\mathcal{G} \subseteq \mathcal{E}$ a goal set and $s \in \mathcal{E}$ a point in the environment. A *search path*, $\pi$, with start point $s$ is a path that starts in $s$ and from which every goal position in $\mathcal{G}$ can be seen. The *search ratio*, $\mathrm{sr}(\pi)$, is defined as

$$\mathrm{sr}(\pi) := \max_{p \in \mathcal{G}} \frac{|\pi(p_\pi)| + |p_\pi p|}{|\,\mathrm{sp}(p)|}\,.$$

[4]Alternatively, we can assume that the cost in the start situation is subsumed by an additive constant in the definition of the competitive factor, see Definition 1.1.

An *optimal search path*, $\pi_{\text{opt}}$, is a search path with a minimum search ratio among all possible paths in the environment. We denote the *optimal search ratio* by $\text{sr}_{\text{opt}}$; that is, $\text{sr}_{\text{opt}} := \text{sr}(\pi_{\text{opt}})$.

In other words, we compare the path walked by a searcher to the shortest path, and take the worst ratio among all possible targets as the search ratio of our search path.

For blind agents, $p = p_\pi$ holds for every $p \in \mathcal{E}$; therefore, the search ratio can be computed as

$$\text{sr}(\pi) := \max_{p \in \mathcal{G}} \frac{|\pi(p)|}{|\,\text{sp}(p)|}\,.$$

The optimal search path seems hard to compute for certain types of environments, so we are interested in finding good approximations for the optimal search path in offline and online scenarios. We say that a strategy, $S$, computes a *$C$-approximation* of the optimal search path, if there are constants $C \geq 1$ and $A \geq 0$, so that $\text{sr}(\pi_S) \leq C \cdot \text{sr}_{\text{opt}} + A$ holds for every path $\pi_S$ computed by $S$.

Remark that this way of measuring performance is one step beyond competitivity. Although the definitions of the search ratio and the competitive factor are quite similar, the concepts are completely different. We no longer give the worst-case ratio between an online solution and the optimal path. Instead, we compare the worst-case ratio of an online solution to the best possible worst-case ratio; in other words, we deal with a ratio of ratios in online settings. Thus, a strategy that approximates the optimal search path up to a constant factor can still be arbitrarily bad in the competitive framework.

In the following, we want to use existing exploration algorithms to approximate the optimal search path. For convenience, we assume that all exploration algorithms always return to the start point. Let $|A|$ denote the length of the path generated by an arbitrary algorithm $A$.

**Definition 4.2** Let *Expl* be an—online or offline—algorithm for the exploration of environments of the given type. *Expl* is called *depth restrictable*, if for every $d \geq 1$ it is possible to modify the algorithm *Expl* to an algorithm *Expl*$(d)$ that explores the environment only up to the depth $d$. That is, the algorithm inspects[5] at least all points of distance $\leq d$—and maybe some more—before it returns to the start point. Let $\textit{Expl}_{\text{opt}}(d)$ denote the corresponding optimal exploration. If there are constants $\beta > 0$

[5] A blind agent has to visit the points, a searcher with vision has to "see" them. We use the term *inspect* to summarize these cases.

and $C_\beta \geq 1$ so that, for every $d \geq 1$, *Expl*$(d)$ is $C_\beta$-competitive with respect to *Expl*$_{\text{opt}}(\beta \cdot d)$—that is,

$$|\mathit{Expl}(d)| \leq C_\beta \cdot |\mathit{Expl}_{\text{opt}}(\beta \cdot d)| \tag{4.1}$$

holds for every environment of the given type—, *Expl* is called *$C_\beta$-depth restrictable*.

Remark that we compare *Expl*$(d)$ to an optimal solution that explores the environment not only up to $d$, but up to $\beta \cdot d$, in contrary to the usual competitive framework. Of course, we prefer depth-restrictable exploration algorithms with $\beta = 1$, but, unfortunately, there are cases where we cannot restrict *Expl* to the depth $d$, because there may be useful shortcuts outside this exploration depth. Even worse, in an online setting it may be impossible to determine which parts of the environment are inside the exploration depth, so we may explore too much of the environment. Thus, it may be easier to find appropriate exploration algorithms for $\beta > 1$. For example, there is an exploration algorithm for graphs by Duncan et al. [49] that is depth restrictable with $\beta = 1 + \alpha$ and $C_\beta = 4 + \frac{8}{\alpha}$ for $\alpha > 0$, see Fleischer et al. [59].

## 4.2 Approximating the Optimal Search Path

We are interested in adequate approximations to the optimal search path, because the optimal search path is hard or even impossible to compute, or the searcher does not know its environment in advance, and, thus, has to approximate the optimal search path online. Of course, a good approximation keeps the approximation factor as small as possible. We give a general approximation framework, and apply this framework to simple polygons.

### 4.2.1 An Approximation Framework

In this section, we assume that there exists an online or offline $C_\beta$-depth-restrictable exploration algorithm, *Expl*, for the given environment. We use *Expl* to approximate the optimal search path.

**Theorem 4.3** *Let Expl be a $C_\beta$-depth-restrictable exploration algorithm for a blind agent. Using Expl, we can compute a $4\beta C_\beta$-approximation of the optimal search path.*

**Proof.** We use the well-known doubling paradigm, see [64, 11] and Section 3.2, and successively apply our given exploration strategy with increasing exploration depth; that is, we successively run *Expl*$(2^i)$, each iteration starting and ending in the start point, $s$.

Now, consider a single iteration of the doubling strategy with search depth $d \geq 1$. Even the optimal search path, $\pi_{\text{opt}}$, has to inspect every possible goal position whose distance from $s$ is at most $d$. Let $p_d$ denote the point with a distance not greater than $d$ that the robot inspects at last while moving along $\pi_{\text{opt}}$. The search ratio of $\pi_{\text{opt}}$ cannot be smaller than the search ratio in $p_d$; this yields

$$\text{sr}_{\text{opt}} \geq \frac{|\pi_{\text{opt}}(p_d)|}{d}\,. \tag{4.2}$$

Moving on the optimal search path, $\pi_{\text{opt}}$, from $s$ to $p_d$, the searcher inspects all points with distance $\leq d$. Therefore, $\pi_{\text{opt}}$ is also a depth-restricted exploration path. However, $\pi_{\text{opt}}$ cannot be shorter than $|\mathit{Expl}_{\text{opt}}(d)|$; otherwise, $\mathit{Expl}_{\text{opt}}(d)$ would not be a shortest exploration tour. Thus, we have

$$|\mathit{Expl}_{\text{opt}}(d)| \leq |\pi_{\text{opt}}(p_d)| + d\,. \tag{4.3}$$

The addend $d$ is necessary, because $\mathit{Expl}_{\text{opt}}(d)$ returns to $s$ whereas $\pi_{\text{opt}}(p_d)$ ends in $p_d$. The searcher is located in $p_d$ while exploring this point because it has no vision. Thus, we can account $|\,\text{sp}(p_d, s)| \leq d$ for the path back to $s$. From Equation 4.2 and Equation 4.3 we get

$$|\mathit{Expl}_{\text{opt}}(d)| \leq d \cdot (\text{sr}_{\text{opt}} + 1)\,. \tag{4.4}$$

Now, our strategy successively applies $\mathit{Expl}(d)$ with increasing exploration depth $d = 2^1, 2^2, 2^3, \ldots$. Similar to the search for a point on a line or on $m$ rays, the worst case for the search ratio of our strategy is achieved if we slightly miss our target in the iteration with exploration depth $d = 2^j$, and inspect almost every point with distance $2^{j+1}$; that is, the distance to the goal is $2^j + \varepsilon$ for a small $\varepsilon > 0$. Altogether, we can bound the search ratio of our approximation strategy by

$$\text{sr}(\pi) \leq \frac{\sum_{i=1}^{j+1} |\mathit{Expl}(2^i)|}{2^j + \varepsilon}\,.$$

*Expl* is $C_\beta$-depth restrictable, so we can apply Equation 4.1

$$\text{sr}(\pi) \leq \frac{C_\beta}{2^j} \cdot \sum_{i=1}^{j+1} |\mathit{Expl}_{\text{opt}}(\beta \cdot 2^i)|\,.$$

Finally, with Equation 4.4 we get

$$\begin{aligned}
\text{sr}(\pi) &\leq \frac{C_\beta}{2^j} \cdot \sum_{i=1}^{j+1} \beta \cdot 2^i \cdot (\text{sr}_{\text{opt}} + 1) \\
&\leq \beta C_\beta \cdot \left(\frac{2^{j+2} - 2}{2^j}\right) \cdot (\text{sr}_{\text{opt}} + 1) \\
&\leq 4\beta\, C_\beta \cdot (\text{sr}_{\text{opt}} + 1)
\end{aligned}$$

Thus, our doubling strategy approximates the optimal search ratio $\mathrm{sr}_{\mathrm{opt}}$ up to a factor of $4\beta C_\beta$. □

A blind searcher has to visit all goal points in a distance $\leq d$, and—particularly—$p_d$, so $\mathrm{sp}(p_d) \leq d$ holds. Unfortunately, we cannot guarantee this for search agents with unlimited vision, because such searchers may see the last point with distance $\leq d$ from somewhere else. We know only that the path back to $s$ is at most as long as the path that the searcher has traveled so far in this iteration; that is, $|\mathrm{sp}(p_d)| \leq |\pi_{\mathrm{opt}}(p_d)|$ holds. Analogously to Equation 4.3 we conclude for a searcher with vision:

$$|\mathit{Expl}_{\mathrm{opt}}(d)| \leq 2 \cdot |\pi_{\mathrm{opt}}(p_d)|\,, \tag{4.5}$$

and with Equation 4.2 we get—compare to Equation 4.4—

$$|\mathit{Expl}_{\mathrm{opt}}(d)| \leq 2d \cdot \mathrm{sr}_{\mathrm{opt}}\,. \tag{4.6}$$

The worst case is achieved for the same setting as described earlier. In contrast to a blind searcher, a searcher with vision has to move to the goal after it is found; thus, the search ratio of our approximation is bounded by

$$\begin{aligned}
\mathrm{sr}(\pi) &\leq \frac{2^j + \varepsilon + \sum_{i=1}^{j+1} |\mathit{Expl}(2^i)|}{2^j + \varepsilon} \\
&\leq 1 + \frac{C_\beta}{2^j} \cdot \sum_{i=1}^{j+1} |\mathit{Expl}_{\mathrm{opt}}(\beta \cdot 2^i)| \\
&\leq 1 + \frac{2C_\beta}{2^j} \cdot \sum_{i=1}^{j+1} \beta 2^i \,\mathrm{sr}_{\mathrm{opt}} \quad \leq \quad 8\beta C_\beta \cdot \mathrm{sr}_{\mathrm{opt}} + 1
\end{aligned}$$

Altogether, we have

**Theorem 4.4** *Let Expl be a $C_\beta$-depth-restrictable exploration algorithm for an agent with vision. Using Expl, we can compute an $8\beta C_\beta$-approximation of the optimal search path.*

Note that a searcher with *limited* vision sees $p_d$ from a distance that is bounded by the range of the vision sensor. Let $r$ denote the maximal vision range, then Equation 4.3 reads

$$|\mathit{Expl}_{\mathrm{opt}}(d)| \leq |\pi_{\mathrm{opt}}(p_d)| + d + r\,.$$

Thus, such a searcher is able to achieve a ratio of $4\beta C_\beta \cdot (\mathrm{sr}_{\mathrm{opt}} + 1) + kr$, where $k$ denotes the number of iterations and depends on the size of $\mathcal{E}$.

### 4.2.2 Searching Simple Polygons

In the previous section, we have seen how to approximate the optimal search path up to a constant factor. The deciding task is to find an appropriate exploration strategy for the given environment that can be turned into a $C_\beta$-depth-restrictable exploration strategy. Now, we apply our framework to simple polygons. We assume that the searcher is equipped with ideal, unlimited vision; that is, the full visibility polygon with respect to the searchers current position is provided. Note that such a searcher is able to inspect potential goal points from a position far away from this point.

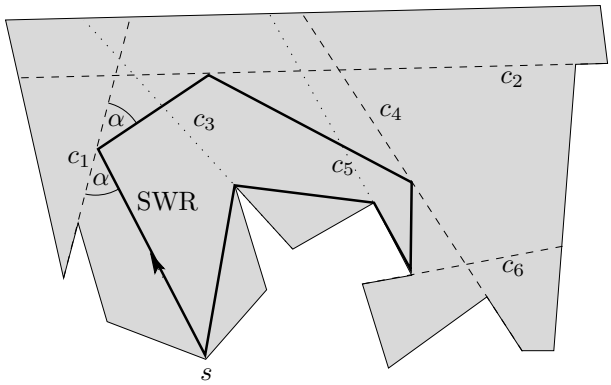



Figure 4.3: A polygon with visibility cuts (dotted), essential cuts (dashed) and Shortest Watchman Route (SWR). The SWR visits the essential cuts using the law of reflections, see $c_1$ for an example.

The *visibility cuts* play a crucial role in the exploration of simple polygons. Consider a reflex vertex, $v$, of $P$. We extend both edges incident to $v$ inside $P$ until they hit the boundary of $P$. Only one of these two extensions is interesting for the visibility; namely, the one that blocks the vision with respect to the start point, $s$. We call this extension a *visibility cut* of $P$ with respect to $s$—or cut for short.[6] A cut, $c_1$, *dominates* another cut, $c_2$, if every path from $s$ to $c_1$ visits $c_2$. Cuts that are not dominated by another cut are called *essential cuts*. Dominated cuts are—so to speak—explored by the way, so we can focus on the essential cuts. Obviously, an exploration path has to visit each essential cut. Figure 4.3 shows an example for a polygon with its visibility cuts (dotted) and essential cuts (dashed). $c_3$ and $c_5$ are not essential, because every path to $c_4$ intersects them.

The optimal offline exploration path is also known as the shortest watchman route, and was first considered by Chin and Ntafos [32]. They gave an $O(n)$-algorithm for shortest watchman routes in rectilinear simple polygons

[6]This definition follows the definition in [84]. Note that there are other definitions for cuts and visibility cuts, see e. g. [33]. However, the definitions of essential cuts are—essentially—the same.

with $n$ vertices. Dror et al. [46] presented an $O(n^3 \log n)$-algorithm for shortest watchman routes in arbitrary simple polygons. An important property of the SWR is that essential cuts are visited using the law of reflection; that is, the angle between the cut and the path segment on which the agent approaches the cut is equal to the angle between the cut and the path segment on which the agent leaves the cut, see Figure 4.3.

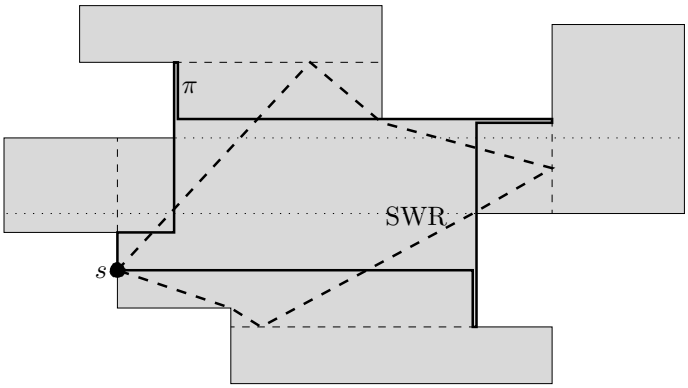



Figure 4.4: Greedy-online exploration and SWR (dashed) in a rectilinear polygon.

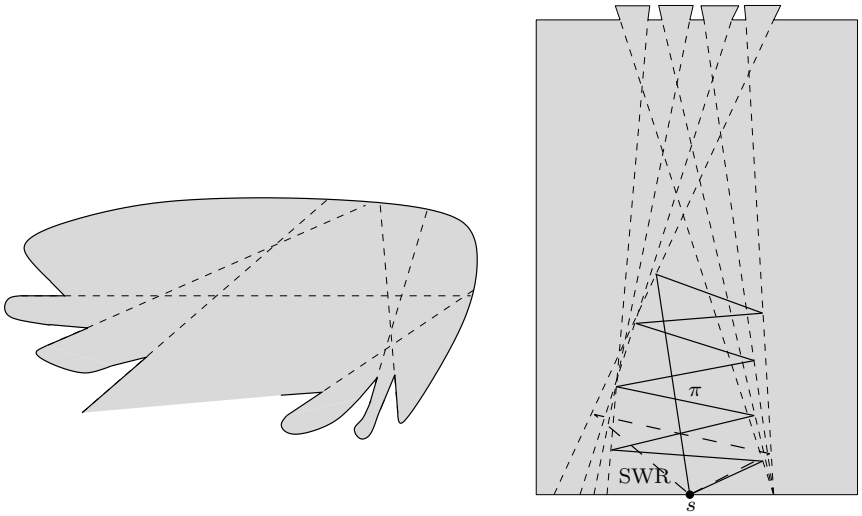
To apply Theorem 4.4 we need a $C_\beta$-depth-restrictable exploration algorithm for simple polygons. More precisely, in a single iteration of the doubling strategy we want to see—at least—all points with a distance smaller than or equal to $d$ from the start point. Let $P(d)$ denote all these points (i. e., $P(d) := \{\, p \in P \mid \mathrm{sp}(s,p) \leq d \,\}$) and $SWR(d)$ the optimal exploration tour for $P(d)$.

Two exploration algorithms for simple polygons are known. The algorithm *greedy online* (*GO*) by Deng, Kameda and Papadimitriou [43] explores a rectilinear polygon by moving successively to the cut of the next reflex vertex in clockwise order along the boundary of the polygon until the whole polygon is explored, see Figure 4.4. Deng et al. showed that *GO* is optimal with respect to the SWR in the Manhatten metric ($L_1$-metric) and $\sqrt{2}$-competitive with respect to the SWR in the Euclidean metric ($L_2$-metric). With this algorithm, we can approximate an optimal search path:

**Theorem 4.5** *There is an online* $8\sqrt{2}$*-approximation of the optimal search path in rectilinear simple polygons for searchers with ideal vision. Further, there is a polynomial time offline* 8*-approximation.*

**Proof.** It is easy to see that *GO* is depth restrictable. We simply ignore vertices farther away than $d$. Because the polygon is simple and rectilinear, both $GO(d)$ and $SWR(d)$ never exceed $P(d)$. Thus, $GO(d) \leq \sqrt{2} \cdot SWR(d)$ holds; that is, *GO* is depth restrictable with $\beta = 1$ and $C_\beta = \sqrt{2}$. With Theorem 4.4 we get our result.

In the offline setting, we can successively apply the algorithm by Chin and Ntafos to $P(d)$. This algorithm yields an optimal exploration path (i. e., $\beta = C_\beta = 1$), so we get an 8-approximation. The running time of this approximation depends on the number of iterations in the doubling strategy, and, in turn, by the distance to the farthest cut. To restrict the number of iterations we simply skip an iteration with exploration depth $d = 2^i$, if there is no (reflex) vertex that would be discovered in this exploration. Thus, we explore at least one new reflex vertex in every iteration and get a maximal number of $k$ iterations for a polygon with $n$ vertices and $k < n$ reflex vertices. Altogether, we achieve a polynomial running time. □




(i) (ii)

Figure 4.5: (i) A corner: A set of essential cuts that intersect each other, (ii) a greedy exploration of reflex vertices is not applicable in nonrectilinear polygons [84].

The correctness proof for *GO* uses some properties of rectilinear polygons. First, the cut of an unexplored vertex is known in advance, and at most two cuts can mutually intersect. Further, the shortest watchman route visits the cuts in the same order as they appear on the polygon's boundary. These properties do no hold in nonrectilinear polygons. Figure 4.5(i) shows a *corner* situation, in which more than two essential cuts intersect each other. In the example shown in Figure 4.5(ii), the greedy exploration—visiting the cuts in same order as the corresponding vertices appear on the polygon's boundary—does not achieve a competitive factor [84].

Nonrectilinear simple polygons can be explored with the 26.5-competitive algorithm *PolyExplore* by Hoffmann, Icking, Klein and Kriegel, see [84]. We give only a very brief insight into the functionality of *PolyExplore*, and refer the reader to [84] for a more comprehensive description. *PolyExplore*

alternately explores groups of right and left reflex vertices[7] to avoid the problem shown in Figure 4.5(ii). The start point of the exploration of a group of vertices is called *stage point*; the initial stage point is $s$.

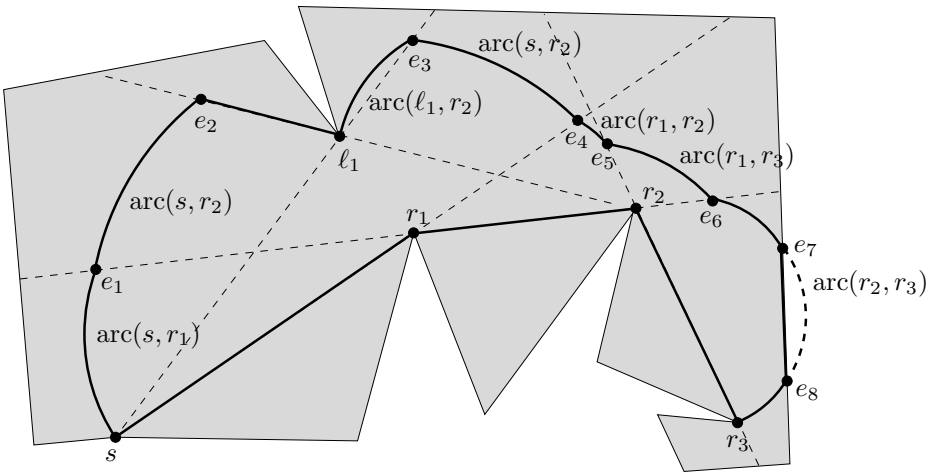



Figure 4.6: Exploring a right reflex vertex with *PolyExplore*.

Moving straight towards the vertices does not achieve a competitive factor, see Icking [86] or Icking et al. [95]; therefore, the robot explores a vertex on a circular arc. The strategy maintains a list, the *target list*, of right vertices that have been seen, but are not fully explored, and whose shortest paths from the current stage point makes only right turns. The target list is sorted in clockwise order along the polygon's boundary. The strategy always aims to reach the cut of the first element in the target list. Figure 4.6 shows an example for the exploration of a right vertex: The robot starts in $s$ and tries to reach $r_1$ on a circular arc. In $e_1$ another right vertex, $r_2$, becomes visible. Because $r_2$ is now the first element in the target list, the robot now follows the arc between $s$ and the new vertex ($\text{arc}(s, r_2)$). There are some events that may occur during a walk on an arc. First, if the sight to the origin of the current arc gets lost, the robots chooses the sight-blocking vertex as new origin. In $e_4$ the sight to $s$ gets lost and the robot follows $\text{arc}(r_1, r_2)$. Further, if the sight to the current target vertex gets lost (see $e_2$), the robot moves a straight line towards the target, until it reaches the vertex that has hidden the target (in our case $\ell_1$). If the robot reaches the polygon's boundary (see $e_7$), it follows a straight line along the boundary until it can continue to follow the current arc.

The preceding procedure explores one essential cut. A group of right vertices may have more than one essential cut, see Figure 4.5(i), so the procedure is called successively until the target list is empty. While a group

[7] A right (left) reflex vertex touches the shortest path tree—the tree of all shortest paths inside the polygon from $s$ to every other vertex—from the right (left, respectively) [84].

of vertices is explored, the algorithm maintains a second list of detected, but unexplored vertices that are not inserted into the target list. The vertices in this list are candidates for subsequent stage points.

We can modify *PolyExplore* to a depth-restricted strategy *PolyExplore*($d$) by ignoring vertices whose distance from the start is greater than $d$. Note that both *PolyExplore*($d$) and $SWR(d)$ may exceed $P(d)$ as shown in Figure 4.7. In (i) *PolyExplore*($d$) explores successively the vertices $v_r$ and $v_\ell$, but $SWR(d)$ visits the cuts outside $P(d)$. In (ii) *PolyExplore*($d$) starts to explore $r_1$. In $e_1$ we discover $r_2$ and continue moving on $\text{arc}(s, r_2)$, until $r_2$ is fully explored in $e_2$. Now, we move on $\text{arc}(e_2, r_3)$. This movement is interrupted in $e_3$, because the sight to $r_3$ gets lost, and leaves $P(d)$ in $e_4$.

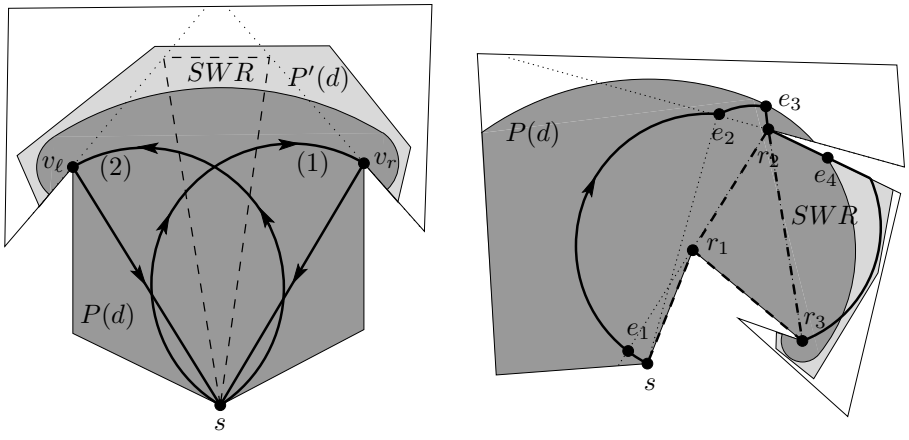



Figure 4.7: Both (i) $SWR(d)$ and (ii) *PolyExplore*($d$) may exceed $P(d)$ (dark gray), but both of them are in $P'(d)$ (light gray).

However, we can enlarge $P(d)$ to $P'(d)$ by a convex region, such that the resulting polygon contains *PolyExplore*($d$) as well as $SWR(d)$, see Figure 4.7. Because we add no reflex vertices, we do not change the paths of $SWR(d)$ and *PolyExplore*($d$), even if the extensions for different parts of $P(d)$ overlap. Thus, the analysis of *PolyExplore* by Hoffmann et al. still holds in $P'(d)$. Therefore, we have

$$\textit{PolyExplore}(d) \leq 26.5 \cdot SWR(d)\,, \tag{4.7}$$

and *PolyExplore* is depth restrictable with $\beta = 1$ and $C_\beta = 26.5$.

**Theorem 4.6** *There is an online* 212*-approximation of the optimal search path in simple polygons for searchers with ideal vision. Further, there is a polynomial time offline* 8*-approximation.*

**Proof.** The factor in the online setting follows from Equation 4.7 and Theorem 4.4. In the offline setting we can—analogously to the rectilinear case—successively apply the algorithm by Dror et al. [46] to $P(d)$ and get an 8-approximation. The polynomial running time follows from the same conclusions as in Theorem 4.5. □

## 4.3 Hard-Searchable Environments

In the previous section, we have seen a general framework for (online) approximations of the optimal search path in environments that are explorable with a competitive, depth-restrictable algorithm.

Now, it is obvious to ask what we can do if there is no constant-competitive online exploration for the given type of environment. Can we find another way to approximate the optimal search path, or can we show that no approximation exists? We call the latter environments *hard searchable*. In the following, we show that polygons with holes are in fact hard searchable. Afterward, we generalize this approach to environments that fulfill a certain condition.

### 4.3.1 Polygons with Holes

It was shown by Albers et al. [5] that there is no exploration algorithm with a constant competitive factor for polygons with (rectilinear) holes by constructing a scene as shown in the upper half of Figure 4.8. This scene consists of $k$ thin rectangles of width $2k$ (spike rectangles) and $k-1$ rectangles of width 1 (base rectangles), for $k \geq 2$. There is a small space between two rectangles that allows the robot to pass, but blocks the sight to the area behind a base rectangle. Behind *one* of the base rectangles we find the same, but scaled construction (recursive subproblem). For the time being, we do not define whether the base rectangles of the subproblem are located on the left side or on the right side. Now, any exploration strategy starting in $s$ has to decide, whether it walks upwards and inspects the base rectangles from the left side, or walks to the right and upwards, inspecting the base rectangles from the right side. In the first case, we fix the base rectangles of the subproblem on the left side, in the second case on the right side. Altogether, the robot's path length is at least $k$ for a single level. We have $k$ subproblems, so $|\pi_{\mathrm{A}}| \in \Omega(k^2)$ holds for every online strategy $A$. On the other hand, the optimal strategy knows the location of the recursive subproblem in advance and directly enters it. The robot leaves the subproblem on the opposite side and is now able to explore the parts behind the remaining base rectangles. Thus, the optimal strategy gets by with a path of length $O(k)$. For $k = \lfloor\sqrt{n}\rfloor$ we get a bound lower of $\Omega(\sqrt{n})$.

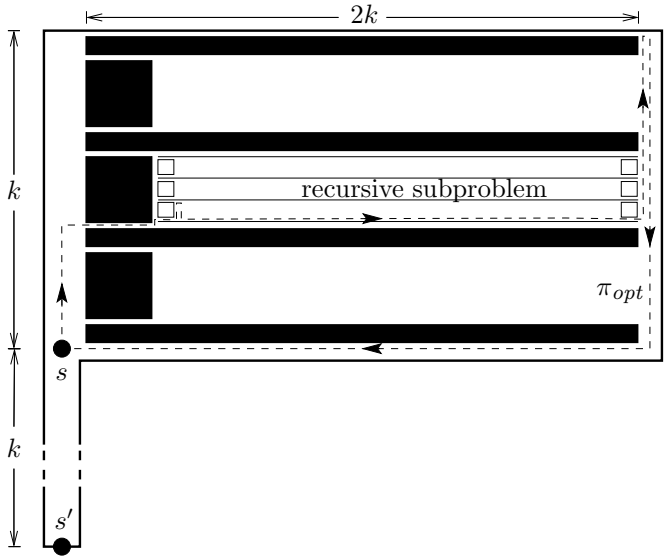



Figure 4.8: Lower bound construction for approximating the optimal search path in polygons with holes (start point $s'$). The upper half shows the lower bound for the exploration task by Albers et al. (start point $s$), and the optimal exploration path $\pi_{\text{opt}}$.

**Theorem 4.7** *There is no online $C$-approximation of the optimal search path for an agent with vision searching in polygons with holes. Further, there is an exponential-time offline 8-approximation in this setting.*

**Proof.** Unfortunately, in the lower bound construction of Albers et al. the optimal exploration paths yields already a bad search ratio. Thus, we enlarge the setting by a thin corridor of length $k$ that leads to the former start point, $s$. Our new start point, $s'$, is located at the end of the new corridor, see Figure 4.8. Now, any point that is not visible from $s'$ is at least $k$ steps away from $s'$; that is, $\text{sp}(s', p) \geq k$ holds for such a point $p$. The optimal exploration path is still never longer than $C \cdot k$ for a constant $C$; thus, the optimal exploration path is a $C$-approximation of the optimal search path. In the new scene, every online exploration algorithm is forced to a path of length in $\Omega(k^2)$. Because every online approximation of the optimal search path is also an online exploration algorithm, there are points that are discovered after walking a path length in $\Omega(k^2)$, although their distance to $s'$ is in $O(k)$. Thus, no online approximation is able to achieve a constant approximation factor.

Finding a shortest watchman route in polygons with holes is NP-hard, see Chin and Ntafos [32]. But we can enumerate all permutations of the essential cuts, compute the watchman route that visits the cuts in this order, and choose the route with minimal length. With Theorem 4.4 we get an 8-approximation, although with exponential running time. □

### 4.3.2 Arbitrary Hard Searchable Environments

We have seen that for environments of a given type there exists an approximation for the optimal search path, if there exists a depth-restrictable, competitive exploration strategy. Further, we have seen that there is no approximation for the optimal search path up to a constant factor in polygons with holes. Now, we want to generalize the latter result; that is, we want to show that—under a certain condition—there is no approximation up to a constant factor, if there is no competitive exploration strategy for environments of the given type.

Usually, the nonexistence of competitive exploration strategies is shown by giving a lower bound—a scenario, in which every exploration strategy is forced to walk a path whose length exceeds the length of the optimal exploration path by more than a constant factor. To transfer such a result to search path approximations, we require that the scenario can be extended around the start point, such that the start point moves further away from the original scenario. We used this technique in the previous section.

**Definition 4.8** Let $\mathcal{E}$ be an environment of arbitrary type, and $s$ be a start point in $\mathcal{E}$. We call $\mathcal{E}$ *$s$-extendable*, if we can enlarge $\mathcal{E}$ locally around the start point; that is, it is possible to choose a new start point, $s'$, outside $\mathcal{E}$, and enlarge $\mathcal{E}$ to $\mathcal{E}'$, such that $s'$ is contained in $\mathcal{E}'$ and every path from $s'$ to a point in $\mathcal{E}$ passes $s$.

**Theorem 4.9** *If there is no constant-competitive online exploration algorithm for environments of a given type, and the corresponding lower bound is $s$-extendable, then there is no competitive online approximation of the optimal search path.*

**Proof.** Let $\mathcal{S}$, $|\mathcal{S}| = n$, be the lower bound construction, $Expl_{\text{opt}}$ denote the optimal exploration algorithm, and $f(n)$ be a function, so that $|Expl_{\text{opt}}| \in O(f(n))$ holds. There is no competitive online exploration, so $|Expl_{\text{A}}| \in \omega(f(n))$ holds for every online algorithm $A$.[8]

Because any online approximation of the optimal search path is also an online exploration strategy, any online approximation strategy in $\mathcal{S}$ can be forced to detect the last point, $p$, after traversing a path with a length in $\omega(f(n))$.

We construct a lower bound, $\mathcal{S}'$, for the online approximation of the search path by placing a new start point, $s'$, *outside* $\mathcal{S}$ with distance $f(n)$ and connecting it to the former start point $s$. In $\mathcal{S}'$ any online search strategy can also be forced to detect the last point, $p$, after moving a path in $\omega(f(n))$, whereas the distance from $p$ to $s'$ is still in $O(f(n))$. Therefore, the search

[8] Whereas $\Omega(g)$ denotes the class of functions growing at least as fast as $g$, $\omega(g)$ denotes the class of functions that grow faster than $g$, see e. g. [187].

ratio of any online search path in $\mathcal{S}'$ is in $\Omega(n)$. On the other hand, the optimal exploration path in $\mathcal{S}'$ is still in $O(f(n))$, yielding—of course—a constant optimal search ratio. Altogether, no constant-competitive online approximation exists. □

## 4.4 Summary

We have seen that there are environments where no online search strategy can achieve a constant competitive factor. Therefore, we use the *search ratio* as a parameter of a given environment that gives a measure for the environment's searchability. A search strategy is considered "good", if it achieves a good approximation of the optimal search ratio; that is, the search ratio of an online strategy is at most a constant factor worse than the optimal search ratio.

We showed that we can use $C_\beta$-depth-restrictable exploration strategies—exploration strategies that can be modified to explore the environment only up to a certain depth while they are still competitive—, to approximate the optimal search path by successively applying the exploration with exponentially increasing exploration depths. For blind agents we showed that there are $4\beta C_\beta$-approximations, for searchers with vision $8\beta C_\beta$-approximations, where $\beta$ and $C_\beta$ are parameters that depend on the modifications to turn an exploration algorithm into a depth-restricted exploration—provided that the environments, $\mathcal{E}$, fulfills $\forall p \in \mathcal{E} : |\operatorname{sp}(s,p)| = |\operatorname{sp}(p,s)|$. We applied our results to simple polygons; applications to trees and graphs can be found in Fleischer et al. [59].

Further, we showed that there is no $C$-approximation of the optimal search path for polygons with holes. The main idea for this proof—enlarging the environment close to the start point—can be generalized for environments that fulfill a certain condition we called $s$-extendable.

If we assume that for every type of environment, for which a constant-competitive exploration strategy is known, there is also a $C_\beta$-depth-restrictable exploration strategy, and for every lower bound there is a $s$-extendable lower bound, we can state the following

**Conjecture 4.10** *For a given type of environments that fulfills $\forall p \in \mathcal{E} : |\operatorname{sp}(s,p)| = |\operatorname{sp}(p,s)|$, there is a constant-competitive online approximation of the optimal search path* ***if and only if*** *there exists a constant-competitive online exploration for environments of this type.*

Proving this conjecture would show a close relation between exploration and searching: We are able to approximate the optimal search path—with other words, we can find a good search strategy—if there is a constant-competitive exploration strategy. And, vice versa, we have no chance to find a good search strategy, if no constant-competitive exploration is possible.

# Chapter 5

# Conclusions

In this work, we discussed several aspects of exploration and searching. We already gave more detailed summaries and pointed out some open problems in Section 2.4.6 (page 69), Section 3.3 (page 104), and Section 4.4. Thus, we review only some items concerning the modeling of robot motion tasks in this chapter.

**Environment**
In Chapter 2, we studied the case of a simple robot moving around a simple type of environment. This view led to clear and easy-to-implement algorithms. Nevertheless, the considered models are not too simple and powerful enough to model certain "real-world" problems like lawn-mowing or cleaning.

**Robot**
In practice, there are no error-free robots. Every type has more or less big errors in driving and sensor readings. Thus, we are interested in giving performance results that consider possible errors, or even design robot models and strategies that take the error into account. In Chapter 3 we studied two examples of robots with errors. For the Pledge algorithm we derived in Section 3.1 a set of sufficient conditions that guarantee a successful execution. In Section 3.2 we gave upper bounds for the error of an robot applying the usual doubling strategy, $f_i = 2^i$. Moreover, we were able to design strategies for searching on the line and on $m$ rays that take the error into account and achieve an optimal competitive factor. These results may justify further work on incorporating error models into theoretically well-understood algorithms that are more complicated and more applicable in practice.

**Costs**

The competitive framework—see Definition 1.1—is a commonly used and normally convenient framework for evaluating online algorithms. Nevertheless, we have seen two examples where the competitive analysis is not appropriate. In Section 2.1 we showed a lower bound of 2 for the exploration of grid polygons with holes. The simple DFS strategy already achieves this factor; thus, DFS is an optimal strategy in the competitive framework. A more rigorous analysis showed that DFS is highly improvable, because it does not take advantage of larger areas in the polygon. Analyzing the problem in a cost model similar to the excess distance ratio—see the paragraph on costs in Chapter 1—led to an algorithm with a better performance than DFS. In this case the competitive framework is quasi not tight enough.

On the other hand, for the search problems we discussed in Chapter 4 the competitive framework is too tight, because no search strategy was able to achieve a constant competitive factor. Thus, we had to find another method to compare search strategies, which led to the concept of search ratios. Approximating the optimal search ratio introduced a kind of "meta-competitity", where we compare two ratios instead of two path lengths. Further, the search ratio reveals a closer relation between exploration and search.

# List of Figures

# Bibliography


[1] H. Abelson and A. A. diSessa. *Turtle Geometry*. MIT Press, Cambridge, 1980.

[2] S. Abramowski and H. Müller. *Geometrisches Modellieren*. BI-Wissenschaftsverlag, Mannheim, 1991.

[3] R. Ahlswede and I. Wegener. *Suchprobleme*. Teubner, Stuttgart, 1979.

[4] S. Albers and M. Henzinger. Exploring unknown environments. In *Proc. 12th Annu. ACM Sympos. Theory Comput.*, pages 416–425, 1997.

[5] S. Albers, K. Kursawe, and S. Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32:123–143, 2002.

[6] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous*. Kluwer Academic Publications, 2003.

[7] D. Angluin, J. Westbrook, and W. Zhu. Robot navigation with distance queries. *Siam J. Comput.*, 30(1):110–144, 2000.

[8] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Comput. Geom. Theory Appl.*, 17:25–50, 2000.

[9] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 2–11, 1996.

[10] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Competitive algorithms for the on-line traveling salesman. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *Lecture Notes Comput. Sci.*, pages 206–217. Springer-Verlag, 1995.

[11] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Inform. Comput.*, 106:234–252, 1993.

[12] M. A. Batalin and G. S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. In *Proc. Internat. Workshop Inform. Proc. Sensor Networks*, pages 376–391, 2003.

[13] M. A. Batalin and G. S. Sukhatme. Efficient exploration without localization. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 2003.

[14] A. Beck and D. J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8:419–429, 1970.

[15] P. Berman. On-line searching and navigation. In A. Fiat and G. Woeginger, editors, *Competitive Analysis of Algorithms*. Springer-Verlag, 1998.

[16] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen, and M. Saks. Randomized robot navigation algorithms. In *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, pages 75–84, 1996.

[17] S. Bespamyatnikh. An $O(n \log n)$ algorithm for the zoo-keeper's problem. *Comput. Geom. Theory Appl.*, 24:63–74, 2003.

[18] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2–3):231–254, 1995.

[19] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM J. Comput.*, 26(1):110–137, Feb. 1997.

[20] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 132–142, 1978.

[21] J. Borenstein and L. Feng. Umbmark: a benchmark test for measuring odometry errors in mobile robots. Technical Report UM-MEAM-94-22, University of Michigan, December 1994.

[22] J. Borenstein and L. Feng. Umbmark: A benchmark test for measuring odometry errors in mobile robots. In *Proc. 1995 SPIE Conf. Mobile Robots*, pages 113–124, 1995.

[23] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, UK, 1998.

[24] P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 23–28, 1992.

[25] J. Boyar and K. S. Larsen. The seat reservation problem. *Algorithmica*, 25:403–417, 1999.

[26] C. Bröcker and A. López-Ortiz. Position-independent street searching. In *Proc. 6th Workshop Algorithms Data Struct.*, volume 1663 of *Lecture Notes Comput. Sci.*, pages 241–252. Springer-Verlag, 1999.

[27] W. Burgard, F. Dellaert, D. Fox, and S. Thrun. Robust monte carlo localization for mobile robots. *Artif. Intell.*, 128:99–141, 2001.

[28] J. Byrne, R. Jarvis, S. Yuta, and A. Zelinsky. Planning paths of complete coverage of an unstructured environment by a mobile robots. In *Internat. Conf. Adv. Robotics*, pages 553–538, 1993.

[29] J. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. In *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 49–60, 1987.

[30] S. Carlsson and H. Jonsson. Computing a shortest watchman path in a simple polygon in polynomial time. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *Lecture Notes Comput. Sci.*, pages 122–134. Springer-Verlag, 1995.

[31] S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete Comput. Geom.*, 22(3):377–402, 1999.

[32] W. Chin and S. Ntafos. Optimum watchman routes. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 24–33, 1986.

[33] W.-P. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete Comput. Geom.*, 6(1):9–31, 1991.

[34] W.-P. Chin and S. Ntafos. The zookeeper problem. *Inform. Sci.*, 63:245–259, 1992.

[35] K. S. Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. In *IEEE Internat. Conf. Robot. Automat.*, pages 2783–2788, 1997.

[36] H. Choset. Coverage for robotics - A survey of recent results. *Ann. Math. Artif. Intell.*, 31:113–126, 2001.

[37] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.

[38] J. Czyzowicz, P. Egyed, H. Everett, D. Rappaport, T. Shermer, D. Souvaine, G. Toussaint, and J. Urrutia. The aquarium keeper's problem. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 459–464, Jan. 1991.

[39] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[40] E. Demaine, A. López-Ortiz, and I. Munro. Robot localization without depth perception. In *Proc. 8th Scand. Workshop Algorithm Theory*, volume 2368 of *Lecture Notes Comput. Sci.*, pages 249–259, 2002.

[41] E. D. Demaine, S. P. Fekete, and S. Gal. Online searching with turn cost. Submitted to Theor. Comput. Sci.

[42] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 298–303, 1991.

[43] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *J. ACM*, 45(2):215–245, 1998.

[44] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32:265–297, 1999.

[45] C. Dienelt. Ein Java Applet zur Exploration gitterförmiger Umgebungen in 3D. Diplomarbeit, Universität Bonn, 2005. http://www.geometrylab.de/Gridrobot3D/.

[46] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *Proc. 35th Annu. ACM Sympos. Theory Comput.*, pages 473–482, 2003.

[47] G. Dudek, E. Milios, and I. M. Rekleitis. Multi-robot collaboration for robust exploration. *Ann. Math. Artif. Intell.*, 31:7–40, 2001.

[48] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. *SIAM J. Comput.*, 27(2):583–604, Apr. 1998.

[49] C. A. Duncan, S. G. Kobourov, and V. S. A. Kumar. Optimal constrained graph exploration. In *Proc. 12th ACM-SIAM Symp. Discr. Algo.*, pages 307–314, 2001.

[50] R. El-Yaniv, R. Kaniel, and N. Linial. Competitive optimal on-line leasing. *Algorithmica*, 25:116–140, 1999.

[51] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6):46–57, 1989.

[52] B. Engels. Navigation in Gitterumgebungen für verteilte Robotersysteme mit eingeschränkter Sensorik. Diplomarbeit, Universität Bonn, August 2005. http://www.geometrylab.de/RacingRobots/.

[53] A. Eubeler. Personal communication, 2004.

[54] A. Eubeler, R. Fleischer, T. Kamphans, R. Klein, E. Langetepe, and G. Trippen. Competitive online searching for a ray in the plane. In *Abstracts 21st European Workshop Comput. Geom.*, pages 107–110, 2005.

[55] H. Everett. Hamiltonian paths in non-rectangular grid graphs. Report 86-1, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1986.

[56] S. Fekete, R. Klein, and A. Nüchter. Online searching with an autonomous robot. In *Proc. 6th Workshop Algorithmic Found. Robot.*, pages 335–350, 2004.

[57] A. Fiat and G. Woeginger, editors. *On-line Algorithms: The State of the Art*, volume 1442 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1998.

[58] R. Fleischer. On the bahncard problem. *Theoretical Computer Science*, 268(1):161–174, October 2001.

[59] R. Fleischer, T. Kamphans, R. Klein, E. Langetepe, and G. Trippen. Competitive online approximation of the optimal search ratio. In *Proc. 12th Annu. European Sympos. Algorithms*, volume 3221 of *Lecture Notes Comput. Sci.*, pages 335–346. Springer-Verlag, 2004.

[60] R. Fleischer, T. Kamphans, R. Klein, E. Langetepe, and G. Trippen. Competitive search ratio of graphs and polygons. In *Abstracts 20th European Workshop Comput. Geom.*, pages 127–130. Universidad de Sevilla, 2004.

[61] R. Fleischer, K. Romanik, S. Schuierer, and G. Trippen. Optimal robot localization in trees. *Information and Computation*, 171:224–247, 2001.

[62] R. Fleischer and G. Trippen. Exploring an unknown graph efficiently. In *Proc. 13th Annu. European Sympos. Algorithms*, volume 3669 of *Lecture Notes Comput. Sci.*, pages 11–22. Springer-Verlag, 2005.

[63] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Comput. Geom. Theory Appl.*, 24:197–224, 2003.

[64] S. Gal. *Search Games*, volume 149 of *Mathematics in Science and Engeneering*. Academic Press, New York, 1980.

[65] S. Gal. Continous search games. In *Search theory: some recent developments.*, volume 112 of *Lecture Notes in pure and applied mathematics*, pages 33–53. Dekker, New York, NY, 1989.

[66] The interactive Geometry-Lab. http://www.geometrylab.de/.

[67] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*, volume 27 of *Discrete Mathematics and Its Applications*. CRC Press LLC, Boca Raton, FL, 2nd edition, 2004.

[68] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, second edition, 1994.

[69] M. Grigni, E. Koutsoupias, and C. H. Papadimitriou. An approximation scheme for planar graph TSP. In *Proc. 36th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 640–645, 1995.

[70] P. Gritzmann. Personal communication via Rolf Klein, 2001.

[71] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 50–63, 1987.

[72] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. *SIAM J. Comput.*, 26(4):1120–1138, Aug. 1997.

[73] D. Halperin, L. E. Kavraki, and J.-C. Latombe. Robotics. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 755–778. CRC Press LLC, Boca Raton, FL, 1997.

[74] D. Halperin, L. E. Kavraki, and J.-C. Latombe. Robot algorithms. In M. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 21, pages 21.1–21.21. CRC Press LLC, 1999.

[75] M. Hammar and B. J. Nilsson. Concerning the time bounds of existing shortest watchman route algorithms. In *Proc. 11th International Symposium on Fundamentals of Computation Theory*, volume 1279 of *Lecture Notes Comput. Sci.*, pages 210–221. Springer-Verlag, Sept. 1997.

[76] M. Hammar, B. J. Nilsson, and S. Schuierer. Parallel searching on $m$ rays. *Comput. Geom. Theory Appl.*, 18:125–139, 2001.

[77] U. Handel, C. Icking, T. Kamphans, E. Langetepe, and W. Meiswinkel. Gridrobot — an environment for simulating exploration strategies in unknown cellular areas. Java Applet, 2000. http://www.geometrylab.de/Gridrobot/.

[78] U. Handel, T. Kamphans, E. Langetepe, and W. Meiswinkel. Polyrobot — an environment for simulating strategies for robot navigation in polygonal scenes. Java Applet, 2002. http://www.geometrylab.de/Polyrobot/.

[79] A. Hemmerling. *Labyrinth Problems: Labyrinth-Searching Abilities of Automata*. B. G. Teubner, Leipzig, 1989.

[80] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.

[81] J. Hertzberg, K. Lingemann, A. Nüchter, and H. Surmann. Fast acquiring and analysis of three dimensional laser range data. In *Proc. 6th Internat. Fall Workshop Vision, Modell., and Visualization*, pages 59 – 66, 2001.

[82] C. Hipke, C. Icking, R. Klein, and E. Langetepe. How to find a point on a line within a fixed distance. *Discrete Appl. Math.*, 93:67–73, 1999.

[83] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. A competitive strategy for learning a polygon. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 166–174, 1997.

[84] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM J. Comput.*, 31:577–600, 2001.

[85] Y. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Comput. Surv.*, 24(3):219–291, 1992.

[86] C. Icking. *Motion and Visibility in Simple Polygons*. PhD thesis, Department of Computer Science, FernUniversität Hagen, 1994.

[87] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring an unknown cellular environment. In *Abstracts 16th European Workshop Comput. Geom.*, pages 140–143. Ben-Gurion University of the Negev, 2000.

[88] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. On the competitive complexity of navigation tasks. In H. Bunke, H. I. Christensen, G. D. Hager, and R. Klein, editors, *Sensor Based Intelligent Robots*, volume 2238 of *Lecture Notes Comput. Sci.*, pages 245–258, Berlin, 2002. Springer.

[89] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring simple grid polygons. In *11th Internat. Comput. Combin. Conf.*, volume 3595 of *Lecture Notes Comput. Sci.*, pages 524–533. Springer, 2005.

[90] C. Icking and R. Klein. Competitive strategies for autonomous systems. In H. Bunke, T. Kanade, and H. Noltemeier, editors, *Modelling and Planning for Sensor Based Intelligent Robot Systems*, pages 23–40. World Scientific, Singapore, 1995.

[91] C. Icking, R. Klein, P. Köllner, and L. Ma. Java applets for the dynamic visualization of voronoi diagrams. In R. Klein, H. W. Six, and L. Wegner, editors, *Computer Science in Perspective: Essays Dedicated to Thomas Ottmann*, volume 2598 of *Lecture Notes Comput. Sci.*, pages 191–205. Springer, Berlin, 2002.

[92] C. Icking, R. Klein, and E. Langetepe. Searching for the kernel of a polygon: A competitive strategy using self-approaching curves. Technical Report 211, Department of Computer Science, FernUniversität Hagen, Germany, 1997.

[93] C. Icking, R. Klein, and E. Langetepe. An optimal competitive strategy for walking in streets. In *Proc. 16th Sympos. Theoret. Aspects Comput. Sci.*, volume 1563 of *Lecture Notes Comput. Sci.*, pages 110–120. Springer-Verlag, 1999.

[94] C. Icking, R. Klein, E. Langetepe, S. Schuierer, and I. Semrau. An optimal competitive strategy for walking in streets. *SIAM J. Comput.*, 33:462–486, 2004.

[95] C. Icking, R. Klein, and L. Ma. How to look around a corner. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 443–448, 1993.

[96] C. Icking, G. Rote, E. Welzl, and C. Yap. Shortest paths for line segments. *Algorithmica*, 10:182–200, 1993.

[97] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.

[98] P. Jaillet and M. Stafford. Online searching. *Operations Research*, 49(4):501–515, July 2001.

[99] B. Kalyanasundaram and K. Pruhs. A competitive analysis of algorithms for searching unknown scenes. *Comput. Geom. Theory Appl.*, 3:139–155, 1993.

[100] B. Kalyanasundaram and K. Pruhs. Constructing competitive tours from local information. *Theoret. Comput. Sci.*, 130:125–138, 1994.

[101] T. Kamphans and R. Klein. Bewegungsplanung für Roboter. Vorlesungsskript, Universität Bonn, Institut für Informatik, 2001.

[102] T. Kamphans and E. Langetepe. Online Bewegungsplanung für Roboter. Vorlesungsskript, Universität Bonn, Institut für Informatik, 2002.

[103] T. Kamphans and E. Langetepe. The Pledge algorithm reconsidered under errors in sensors and motion. In *Proc. of the 1th Workshop on*

*Approximation and Online Algorithms*, volume 2909 of *Lecture Notes Comput. Sci.*, pages 165–178. Springer, 2003.

[104] T. Kamphans and E. Langetepe. Finding a door along a wall with an error afflicted robot. In *Abstracts 20th European Workshop Comput. Geom.*, pages 143–146. Universidad de Sevilla, 2004.

[105] T. Kamphans and E. Langetepe. On optimizing multi-sequence functionals for competitive analysis. In *Abstracts 21st European Workshop Comput. Geom.*, pages 111–114, 2005.

[106] T. Kamphans and E. Langetepe. Optimal competitive online ray search with an error-prone robot. In *Proc. 4th Internat. Workshop Efficient Experim. Algorithms*, volume 3503 of *Lecture Notes Comput. Sci.*, pages 593–596. Springer, 2005.

[107] T. Kamphans and E. Langetepe. Optimal competitive online ray search with an error-prone robot. Technical Report 003, Department of Computer Science I, University of Bonn, 2005. http://web.informatik.uni-bonn.de/I/publications/kl-ocolr-05t.pdf.

[108] M.-Y. Kao, Y. Ma, M. Sipser, and Y. Yin. Optimal constructions of hybrid algorithms. *J. Algor.*, 29:142–164, 1998.

[109] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Inform. Comput.*, 133(1):63–79, 1996.

[110] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. Tech. Report UU-CS-1994-32, Dept. Comput. Sci., Utrecht Univ., P.O.Box 80.089, 3508 TB Utrecht, The Netherlands, August 1994.

[111] K. Kedem and M. Sharir. An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space. *Discrete Comput. Geom.*, 5:43–75, 1990.

[112] K. Kedem, M. Sharir, and S. Toledo. On critical orientations in the Kedem-Sharir motion planning algorithm for a convex polygon in the plane. *Discrete Comput. Geom.*, 17:227–240, 1997.

[113] R. Klein. Walking an unknown street with bounded detour. *Comput. Geom. Theory Appl.*, 1:325–351, 1992.

[114] R. Klein. *Algorithmische Geometrie*. Addison-Wesley, Bonn, 1997.

[115] R. Klein. *Algorithmische Geometrie*. Springer, Heidelberg, 2nd edition, 2005.

[116] J. M. Kleinberg. On-line search in a simple polygon. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 8–15, 1994.

[117] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1st edition, 1968.

[118] E. Koutsoupias, C. H. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Proc. 23th Internat. Colloq. Automata Lang. Program.*, volume 1099 of *Lecture Notes Comput. Sci.*, pages 280–289. Springer, 1996.

[119] B. J. Kuipers and Y.-T. Byun. A robust qualitative method for spatial learning in unknown environments. In *Proc. 7th Nat. Conf. Artif. Intell.* Morgan Kaufman, 1988.

[120] B. J. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *J. Robot. Auton. Syst.*, 8:47–63, 1991.

[121] L. Kusch and H.-J. Rosenthal. *Mathematik Teil 3: Differentialrechnung*. Giradet, 2nd edition, 1970.

[122] E. Langetepe. *Design and Analysis of Strategies for Autonomous Systems in Motion Planning*. PhD thesis, Department of Computer Science, FernUniversität Hagen, 2000.

[123] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[124] S. Laubach and J. Burdick. RoverBug: Long range navigation for mars rovers. In P. Corke and J. Trevelyan, editors, *Proc. 6th Int. Symp. Experimental Robotics*, volume 250 of *Lecture Notes in Control and Information Sciences*, pages 339–348. Springer, 1999.

[125] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. to appear.

[126] S. M. LaValle, S. Rajko, and S. Sachs. Visibility-based pursuit-evasion in an unknown planar environment. *Internat. J. Robot. Res.*, 23:3–26, 2004.

[127] D. Lee and M. Recce. Quantitative evaluation of the exploration strategies of a mobile robot. *Internat. J. Robot. Res.*, 16(4):413–447, 1997.

[128] A. López-Ortiz. *On-line target searching in bounded and unbounded domains*. PhD thesis, Univ. Waterloo, Waterloo, Canada, 1996.

[129] A. López-Ortiz. Algorithmic foundations of the internet. *SIGACT News*, 36:45–62, 2005.

[130] A. López-Ortiz and S. Schuierer. Simple, efficient and robust strategies to traverse streets. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 217–222, 1995.

[131] A. López-Ortiz and S. Schuierer. The ultimate strategy to search on $m$ rays? *Theor. Comput. Sci.*, 261(2):267–295, 2001.

[132] A. López-Ortiz and S. Schuierer. Searching and on-line recognition of star shaped polygons. *Inform. Comput.*, 185:66–88, 2003.

[133] A. López-Ortiz and S. Schuierer. Online parallel heuristics, processor scheduling, and robot searching under the competitive framework. *Theoret. Comput. Sci.*, 310:527–537, 2004.

[134] A. López-Ortiz and G. Sweet. Parallel searching on a lattice. In *Proc. 13th Canad. Conf. Comput. Geom.*, pages 125–128, 2001.

[135] V. J. Lumelsky, S. Mukhopadhyay, and K. Sun. Dynamic path planning in sensor-based terrain acquisition. *IEEE Trans. Robot. Autom.*, 6(4):462–472, Aug. 1990.

[136] V. J. Lumelsky and T. Skewis. Incorporating range sensing in the robot navigation function. *IEEE Trans. Syst. Man Cybern.*, 20(5):1058–1069, 1990.

[137] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[138] V. J. Lumelsky and S. Tiwari. An algorithm for maze searching with azimuth input. In *Proc. 1994 Internat. Conf. Robot. Automat.*, pages 111–116, 1994.

[139] Macromedia. Flash MX. http://www.macromedia.com/software/flash/ Date of access: 04/12/2006.

[140] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric $k$-MST problem. In *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, pages 402–408, 1996.

[141] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 445–466. CRC Press LLC, Boca Raton, FL, 1997.

[142] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[143] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 116–121, 1985.

[144] H. Noborio and T. Yoshioka. On the sensor-based navigation under position and orientation errors. *J. Automatisierungstechnik*, 48:281–288, 2000.

[145] H. Noborio, T. Yoshioka, and T. Hamaguchi. On-line deadlock-free path-planning algorithms by means of a sensor-feedback tracing. In *Proc. IEEE Int. Conf. Systems, Man, Cybernetics*, pages 1291–1296, 1995.

[146] H. Noborio, T. Yoshioka, and T. Hamaguchi. On-line deadlock-free path-planning algorithms in the presence of a dead reckoning error. In *Proc. IEEE Int. Conf. Systems, Man, Cybernetics*, pages 483–488, 1995.

[147] S. Ntafos. The robber route problem. *Inform. Process. Lett.*, 34(2):59–63, Mar. 1990.

[148] S. Ntafos. Watchman routes under limited visibility. *Comput. Geom. Theory Appl.*, 1(3):149–170, 1992.

[149] C. Ó'Dúnlaing and C. K. Yap. A "retraction" method for planning the motion of a disk. *J. Algorithms*, 6:104–111, 1985.

[150] J. O'Rourke. *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[151] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2nd edition, 1998.

[152] M. H. Overmars. A random approach to motion planning. Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, Oct. 1992.

[153] M. H. Overmars. Recent developments in motion planning. Tech. Report UU-CS-2002-004, Inst. of Inform. and Comput. Sci., Utrecht Univ., P.O.Box 80.089, 3508 TB Utrecht, The Netherlands, 2002.

[154] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoret. Comput. Sci.*, 84(1):127–150, 1991.

[155] S. Rajko and S. M. La Valle. A pursuit-evasion bug algorithm. In *Proc. IEEE Conf. Robotics Automat.*, 2001.

[156] A. Randolph. Ricochet robots. Board Game, german edition by Abacus Games, Dreieich, 1999.

[157] N. S. V. Rao, S. S. Iyengar, B. J. Oommen, and R. L. Kashyap. On terrain model acquisition by a point robot amidst polyhedral obstacles. *Internat. J. Robot. Autom.*, 4(4):450–455, 1988.

[158] N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar. Robot navigation in unknown terrains: introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.

[159] J.-R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry.* North-Holland, Amsterdam, 2000.

[160] A. Sankaranarayanan and M. Vidyasagar. Path planning for moving a point object amidst unknown obstacles in a plane: A new algorithm and a general theory for algorithm developments. In *Proceedings of 1990 IEEE Conf. on Decision and Control*, pages 1111–1119, 1990.

[161] S. Schuierer. On-line searching in simple polygons. In H. Christensen, H. Bunke, and H. Noltemeier, editors, *Sensor Based Intelligent Robots*, volume 1724 of *LNAI*, pages 220–239. Springer Verlag, 1997.

[162] S. Schuierer. Searching on $m$ bounded rays optimally. Technical Report 112, Institut für Informatik, Universität Freiburg, Germany, 1998.

[163] S. Schuierer. A lower bound for randomized searching on $m$ rays. In R. Klein, H. W. Six, and L. Wegner, editors, *Computer Science in Perspective: Essays Dedicated to Thomas Ottmann*, volume 2598 of *Lecture Notes Comput. Sci.*, pages 264–277. Springer-Verlag, Berlin, 2003.

[164] S. Schuierer and I. Semrau. An optimal strategy for searching in unknown streets. In *Proc. 16th Sympos. Theoret. Aspects Comput. Sci.*, volume 1563 of *Lecture Notes Comput. Sci.*, pages 121–131. Springer-Verlag, 1999.

[165] J. T. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artif. Intell.*, 37:157–169, 1988.

[166] J. T. Schwartz and M. Sharir. Algorithmic motion planning in robotics. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, pages 391–430. Elsevier, Amsterdam, 1990.

[167] J. T. Schwartz and C. Yap. *Advances in Robotics Vol. I: Algorithmic and geometric aspects of robotics*. Lawrence Erlbaum Associates, 1987.

[168] C. E. Shannon. Presentation of a maze solving machine. In H. von Foerster, M. Mead, and H. L. Teuber, editors, *Cybernetics: Circular, Causal and Feedback Mechanisms in Biological and Social Systems, Transactions Eighth Conference, 1951*, pages 169–181, New York, 1952. Josiah Macy Jr. Foundation. Reprint in [169].

[169] C. E. Shannon. Presentation of a maze solving machine. In N. J. A. Sloane and A. D. Wyner, editors, *Claude Shannon: Collected Papers*, volume PC-03319. IEEE Press, 1993.

[170] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 40, pages 733–754. CRC Press LLC, Boca Raton, FL, 1997.

[171] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

[172] Sick AG. Productinformation laser scanner. http://www.sick.de/de/products/categories/safety/espe/laserscanner/de.html, Date of access: 04/12/2006.

[173] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.

[174] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical Report CMU-RI-TR-93-20, Carnegie Mellon University Robotics Institute, August 1993.

[175] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proc. IEEE Internat. Conf. Robot. Automat.*, pages 3310–3317, 1994.

[176] Sun Microsystems. Java development kit. http://java.sun.com/ Date of access: 04/12/2006.

[177] P. Švestka and M. H. Overmars. Motion planning for car-like robots using a probabilistic learning approach. Tech. Report UU-CS-1994-33, Dept. Comput. Sci., Utrecht Univ., P.O.Box 80.089, 3508 TB Utrecht, The Netherlands, August 1994.

[178] X. Tan. Fast computation of shortest watchman routes in simple polygons. *Inform. Process. Lett.*, 77:27–33, 2001.

[179] X. Tan and T. Hirata. Constructing shortest watchman routes by divide-and-conquer. In *Proc. 4th Annu. Internat. Sympos. Algorithms Comput.*, volume 762 of *Lecture Notes Comput. Sci.*, pages 68–77. Springer-Verlag, 1993.

[180] X. Tan and T. Hirata. Shortest safari routes in simple polygon. In *Proc. 5th Annu. Internat. Sympos. Algorithms Comput.*, volume 834 of *Lecture Notes Comput. Sci.*, pages 523–531. Springer-Verlag, 1994.

[181] X. Tan, T. Hirata, and Y. Inagaki. Corrigendum to "an incremental algorithm for constructing shortest watchman routes". *Internat. J. Comput. Geom. Appl.*, 9(3):319–323, 1999.

[182] X. H. Tan, T. Hirata, and Y. Inagaki. An incremental algorithm for constructing shortest watchman routes. *Internat. J. Comput. Geom. Appl.*, 3(4):351–365, 1993.

[183] C. J. Taylor and D. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. *IEEE Trans. Robot. Autom.*, 14:417–427, 1998.

[184] G. Trippen. Online robot exploration – a survey. Unpublished manuscript, 2002.

[185] C. Umans and W. Lenhart. Hamiltonian cycles in solid grid graphs. In *Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 496–507, 1997.

[186] J. R. VanderHeide and N. S. V. Rao. Terrain coverage of an unknown room by an autonomous mobile robot. Technical Report ORNL/TM-13117, Oak Ridge National Laboratory, 1995.

[187] I. Wegener. *Effiziente Algorithmen für grundlegende Funktionen.* Teubner, 1998.

[188] E. W. Weisstein. Simple polyhedron.
http://mathworld.wolfram.com/SimplePolyhedron.html
Date of access: 04/12/2006.

[189] B. Yamauchi, A. Schultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In *Proc. IEEE Internat. Conf. Robot. Autom.*, 1998.

[190] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transact. Robot. Automat.*, 8(6):707–717, 1992.

# Index

**E**



**F**



**G**



**H**



**I**



**J**



**K**

## L



## M



## N



## O

**P**



**R**



**S**



**T**

**U**



**V**



**W**



**Y**



**Z**

**Curriculum Vitae**

| | |
|---|---|
| Name: | Thomas Kamphans |
| Date of birth: | May 19, 1970 |
| Place of birth: | Unna, Germany |
| School: | 1976 – 1980<br>Katharinen-Grundschule Unna |
| | 1980 – 1989<br>Geschwister-Scholl-Gymnasium Unna |
| Graduation: | May 08, 1989<br>Abitur |
| Studies: | 1990 – 1998<br>University of Dortmund<br>Subject: Computer science<br>Subsidiary subject: Electrical engineering |
| Final degree: | September 08, 1993<br>Vordiplom (intermediate examination) |
| | September 07, 1998<br>Diplom-Informatiker |
| Occupational Activities: | October 1998 – July 2000<br>Research Associate, University of Hagen |
| | Since August 2000<br>Research Associate, University of Bonn |